



# POLITECNICO

## MILANO 1863

Scuola di Ingegneria Industriale e dell'Informazione  
Dipartimento di Elettronica, Informazione e Bioingegneria

Corso di Laurea Magistrale in Computer Science and Engineering

---

Privacy-Preserving Convolutional Neural Networks  
through Homomorphic Encryption

Master Thesis of:  
**Carmen Barletta**  
Matr. 877129

Advisor:  
**Prof. Manuel Roveri**  
Co-Advisor:  
Simone Disabato

Academic Year 2017–2018



---

# Ringraziamenti

*Questi cinque anni universitari si concludono con tante consapevolezze in più. Una e la più preziosa è che non importa quante difficoltà dovrò ancora affrontare, perché so che al mio fianco avrò sempre sei persone straordinarie pronte a sostenermi e a farmi capire quanto sono fortunata.*

*Ringrazio Matteo, una persona speciale, che soprattutto in questi ultimi mesi ha dimostrato di avere un pazienza infinita, lavorando anche nel tempo libero solo per non farmi sentire l'unica sfigata, vestendo i panni del coach armato di "bastone e carota" per spronarmi e consolarmi in ogni momento.*

*Ringrazio Mamma e Papà che hanno costantemente vegliato su di me e non si sono mai stancati di dispensare importantissimi consigli infondendomi coraggio e sicurezza.*

*Ringrazio Giò che è sempre stata un esempio da seguire per me, la roccia su cui contare e la sorella che mi ha sempre dato uno spunto per crescere e sognare.*

*Ringrazio Stè per tutte le chiacchierate, i momenti spensierati, il sorriso e gli abbracci che ha saputo donarmi incondizionatamente.*

*Ringrazio Annalisa, l'amica di una vita, per non avermi mai chiuso la porta in faccia soprattutto quando ne avevo più bisogno.*

*Questo traguardo lo dedico a voi perché ve lo meritate e perché spero che insieme ne raggiungeremo tanti altri.*

*Grazie,*

*Carmen*



---

# Abstract

In the last few years Machine Learning techniques have become popular thanks to their ability to solve complex problems. A popular machine-learning application paradigm considers *Software as a Service*, that is a software distribution model in which a third-party provider, usually a cloud server, hosts applications (i.e., machine learning algorithms) and makes them available to customers over the Internet. On the one hand this cloud-based machine-learning paradigm is useful since it is usually fast and scalable, thus it allows to reach a big number of users, but, on the other hand, it poses severe issues in terms of privacy because analyzed data, on the untrusted cloud server, might be sensible data. This issue is also particularly relevant after the recent approval of the European's General Data Protection Regulation. This thesis provides a methodological solution about how to design a privacy-preserving machine-learning system based on Homomorphic Encryption (HE). More precisely, this work focuses on Convolutional Neural Networks (CNNs) and presents both a methodology and a library to convert a pre-trained CNN to a privacy-preserving CNN able to process encrypted data, by employing the Brakerski, Fan and Vercauteren (BFV) HE scheme. Moreover, this thesis provides a mathematical formulation for the problem of finding the best encryption parameters for the BFV encryption scheme as well as a heuristic, based on a binary search algorithm, to solve the problem of the parameters' choice, tailored on the privacy-preserving CNN. The proposed heuristic aims to find the best transformation for the given CNN. The experimental results show that the proposed methodology is able to operate on CNNs characterized by different architectures and processing pipelines. The results also prove that privacy-preserving deep learning is possible at the cost of a small loss in accuracy and slower predictions.



---

# Estratto

Negli ultimi anni le tecniche di Machine Learning si sono diffuse ampiamente poiché esse permettono di risolvere problemi complessi. Uno dei modelli di machine-learning più comunemente usati è il Software as a Service. Si tratta di un modello di distribuzione del software nel quale un terzo, solitamente un server cloud, ospita le applicazioni (ovvero, gli algoritmi di machine learning) e le rende fruibili agli utenti attraverso Internet. Da un lato questa tecnica di machine learning basata sul cloud risulta utile poiché è solitamente veloce e scalabile, quindi permette di raggiungere un gran numero di utenti, ma, dall'altro lato pone seri problemi in termini di privacy, poiché i dati analizzati sul server, considerato inaffidabile, potrebbero essere dati sensibili. Questo aspetto negativo è particolarmente rilevante soprattutto alla luce della recente approvazione dell'European's General Data Protection Regulation. Questa tesi fornisce una soluzione metodologica alla progettazione di un sistema di machine-learning in grado di preservare la privacy dei dati analizzati, detto privacy-preserving, attraverso l'impiego della crittografia omomorfa. Più precisamente, questo lavoro si focalizza sulle Reti Neurali Convolutionali (CNNs) e presenta sia una metodologia che una libreria per convertire una rete neurale già addestrata in una rete in grado di preservare la privacy, utilizzando lo schema omomorfo di Brakerski, Fan and Vercauteren (BFV). Inoltre, questo lavoro propone una formulazione matematica relativa al problema di trovare i migliori parametri di crittografia per lo schema omomorfo BFV, nonché un'euristica, basata sull'algoritmo di ricerca binaria, per risolvere il problema della scelta dei parametri considerando una specifica privacy-preserving CNN. L'euristica proposta mira quindi a trovare la miglior trasformazione per la data rete convoluzionale. I risultati sperimentali mostrano che la metodologia proposta è in grado di funzionare su CNNs caratterizzate da differenti architetture e pipelines di elaborazione. I risultati

---

provano anche che è possibile preservare la privacy dei dati analizzati con algoritmi di deep learning, pagando un piccolo prezzo in termini di perdita di accuratezza e tempi di predizione più lenti.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem and Motivations . . . . .	1
1.1.1	The EU-GDPR . . . . .	2
1.2	Goals and Results . . . . .	4
1.3	Thesis Structure . . . . .	5
<b>2</b>	<b>Background</b>	<b>7</b>
2.1	Homomorphic Encryption . . . . .	7
2.1.1	Classification of Homomorphic Encryption Schemes . . . . .	8
2.1.2	Brief History of Homomorphic Encryption Schemes . . . . .	10
2.1.3	Limitations of Homomorphic Encryption . . . . .	11
2.2	Brakerski/Fan-Vercauteren scheme (BFV) . . . . .	14
2.2.1	Ring Definition . . . . .	14
2.2.2	Basic Notation . . . . .	14
2.2.3	RLWE Problem . . . . .	15
2.2.4	Encryption Scheme . . . . .	16
2.2.5	Noise . . . . .	17
2.2.6	Operations and Noise Growth . . . . .	19
2.2.7	Operations with plaintext . . . . .	21
2.2.8	Encoding . . . . .	24
2.3	Convolutional Neural Networks: A Brief Overview . . . . .	26
2.3.1	Approximations For Homomorphic Encryption . . . . .	29
2.3.2	Convolutional Neural Networks In Literature . . . . .	29
<b>3</b>	<b>State of the Art</b>	<b>31</b>
3.1	Multiple Data Providers . . . . .	31

3.2	Single Data Provider . . . . .	34
3.3	Other related works . . . . .	36
<b>4</b>	<b>Parameters Estimation Methodology</b>	<b>39</b>
4.1	Encryption Parameters . . . . .	39
4.2	Problem Formulation . . . . .	42
4.3	Optimization Problem . . . . .	43
4.4	Heuristic: Binary Search Of Plaintext Modulus . . . . .	47
<b>5</b>	<b>Practical Considerations About The Proposed Methodology</b>	<b>51</b>
5.1	Simple Encrypted Arithmetic Library (SEAL 2.3.1) . . . . .	51
5.2	Practical Encryption Parameters . . . . .	52
5.3	Practical Optimization Problem . . . . .	52
<b>6</b>	<b>CrCNN:A CNN Library Based On Homomorphic Encryption</b>	<b>55</b>
6.1	Structure And Main Functionalities . . . . .	55
<b>7</b>	<b>Experimental Results</b>	<b>59</b>
7.1	Experimental Setup . . . . .	59
7.2	MNIST Dataset . . . . .	60
7.3	Heuristic comparisons and metrics . . . . .	60
7.4	Case Study 1: 9-Layers CNN . . . . .	63
7.4.1	STEP 1: Approximation . . . . .	64
7.4.2	STEP 2: Encoding . . . . .	64
7.4.3	STEP 3: Testing . . . . .	64
7.4.4	Heuristic metrics and comparison results . . . . .	65
7.4.5	PARTIAL Binary Search Results . . . . .	67
7.4.6	Analysis Of The Suboptimal CNN . . . . .	68
7.4.6.1	Timings and noise . . . . .	68
7.4.6.2	Accuracy Tracking Through Transformation's Steps . . . . .	70
7.5	Case Study 2: 6-Layers CNN . . . . .	70
7.5.1	STEP 1: Approximation . . . . .	71
7.5.2	STEP 2: Encoding . . . . .	71
7.5.3	STEP 3: Testing . . . . .	71
7.5.4	Heuristic metrics and comparison results . . . . .	72

## CONTENTS

---

7.5.5	Analysis Of The Suboptimal CNN . . . . .	74
7.5.5.1	Timings and noise . . . . .	74
7.5.5.2	Accuracy Tracking Through Transformation's Steps . . . . .	75
<b>8</b>	<b>Conclusions and Future Works</b>	<b>77</b>
8.1	Conclusions . . . . .	77
8.2	Future Works . . . . .	78



# List of Figures

3.1	Gradients-encrypted Asynchronous SGD for privacy-preserving deep learning, with a curious cloud server and N honest participants. [56]	34
4.1	Transformation steps of the pre-trained CNN $\check{\Phi}$ .	43
4.2	General structure of a CNN $\Phi$ and below its transformation in $\tilde{\Phi}_{n,q,t}$ in the encryption world accomplished by function $f$ .	45
4.3	Binary Search Cases	48
6.1	UML of Layers' inheritance structure	57
7.1	Maximum and average plain modulus found by the Binary Search for increasing number of images tested ( $K$ )	68



# List of Tables

4.1	Notation used . . . . .	40
4.2	Default pairs (n, q) for 128-bit, 192-bit, and 256-bit $\lambda$ -security levels. . . . .	41
5.1	Notation used in SEAL . . . . .	52
6.1	Main datatypes of CrCNN . . . . .	56
7.1	Column Binary Search shows the encoded 9-layers CNN performances at the varying of the plaintext modulus $t$ . Each $t$ is obtained with a different number of images tested in the heuristic. Column SEAL tool shows the results of the SEAL automatic encryption parameters selection tool for the same 9-layers CNN. . . . .	65
7.2	Testing times of the suboptimal 9-layers $\tilde{\Phi}_{n=4096, q=\{q_1, q_2\}, t=2^{29}}$ and noise budget consumption in the input data (encrypted with the same parameters) through layers . . . . .	69
7.3	9-layers CNN's accuracy mutation . . . . .	70
7.4	Column Binary Search shows the encoded 6-layers CNN performances at the varying of the plaintext modulus $t$ . Each $t$ is obtained with a different number of images tested in the heuristic. Column SEAL tool shows the results of the SEAL automatic encryption parameters selection tool for the same 6-layers CNN. . . . .	72
7.5	Testing times of the suboptimal 6-layers $\tilde{\Phi}_{n=2048, q, t=2^{16}}$ and noise budget consumption in the input data (encrypted with the same parameters) through layers . . . . .	74
7.6	6-layers CNN's accuracy mutation . . . . .	75



## List of Algorithms

4.1	Plaintext Modulus Binary Search . . . . .	49
4.2	Test plaintext modulus . . . . .	50



# Nomenclature

AI	Artificial Intelligence
BFV	Brakerski/Fan-Vercauteren
BGN	Boneh-Goh-Nissim
CNN	Convolutional Neural Network
CPA	Chosen Plaintext Attacks
ELM	Extreme Learning Machines
FHE	Fully Homomorphic Encryption
HE	Homomorphic Encryption
NB	Noise Budget
NTT	Number-Theoretic Transform
RLWE	Ring Learning With Error
SEAL	Simple Encrypted Arithmetic Library
SHE	Somewhat Homomorphic Encryption
SIMD	Single Instruction Multiple Data



# Chapter 1

## Introduction

### 1.1 Problem and Motivations

The early birth of Machine Learning<sup>1</sup> and artificial neural networks<sup>2</sup> can be positioned in 1943, when a neurophysiologist and a mathematician co-wrote a paper to illustrate how to model a neural network with electrical circuits. This work opened the way to the possibility of describing computers as systems that are able to learn from their own experience and solve complex problems in different situations, abilities that were previously thought to be unique to mankind. Since then, machine learning techniques have become more and more popular moving out of research environments and coming into everybody's lives with applications across industries, also thanks to the availability of huge amounts of data, coupled with an increase in processing power and access to cheaper and greater storage capacity. A popular machine-learning application paradigm considers *Software as a Service* (SaaS), that is a software distribution model in which a third-party provider, usually a cloud server, hosts applications (i.e., machine learning algorithms) and makes them available to different customers over the Internet. This cloud-based machine learning technique is useful since it is usually fast, scalable and easy-to-use, thus allowing to reach a big number of users. This model enables to collect and analyze many data that are not property of the cloud server, but of some data providers, and to use them for predictions, image

---

<sup>1</sup>It is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed.

<sup>2</sup>It is a type of learning algorithm that is vaguely inspired by biological neural networks.

and speech recognition and others useful tasks. However, in most of cases these data are sensible data and this places issues in terms of privacy implications, since the server is an untrusted party considered as an "honest but curious sever", that handles computations correctly but tries to capture informations from analyzed data. This problem has become even more relevant after the approval of the European's **General Data Protection Regulation (GDPR)**. The answer to the question as to whether it is possible to use machine learning techniques and protect people's data while doing so, is yes. It is both possible and necessary in order to safeguard fundamental personal data protection rights. The field of cryptography and in particular of Homomorphic Encryption (HE) offers promising possibilities. By employing a Homomorphic Encryption scheme, a data owner can encrypt its data, with its public key, before to send them to a cloud service that hosts the algorithm to compute. The encryption ensures that the data remain confidential since the cloud has not access to the secret key needed to decrypt the data, but at the same time, it is able to perform the computations required on the encrypted data, make predictions, and return them to the data owner in an encrypted form. Thus, only the data owner in the end will be able to decrypt the prediction and obtain its results. Therefore the cloud service does not gain any information about the raw data nor about the prediction it made.

### 1.1.1 The EU-GDPR

The necessity to take into account privacy when dealing with machine learning has been strengthened by the entry into force on 24 May 2016 of the **EU's General Data Protection Regulation (GDPR)**[1]. The GDPR replaces the Data Protection Directive 95/46/EC and was designed to harmonize data privacy laws across Europe, to protect and empower all EU citizens data privacy and to reshape the way organizations across the region approach data privacy. The GDPR has a broad territorial scope. It applies not only to all organizations established in the EU that handles personal data but also to any non-EU established organization that processes personal data of individuals who are in the EU in order to offer them goods or services. The GDPR aims to protect personal data at all stages of data processing and it identifies two different entities that both have obligations: data controllers and data processors. The provisions of the GDPR govern the data controller's duties and the rights of the data subject when personal information is processed.

The GDPR therefore applies when artificial intelligence <sup>3</sup> (AI) is *under development* with the help of personal data, and also when it is used to *analyze* or *reach decisions* about individuals, as for example in Convolutional Neural Networks (CNNs) that analyze sensible images.

It is important to clarify what one means when he speaks about *personal data*. Personal data means any information relating to an identified or identifiable natural person (GDPR Article 4 (1)). The data may be directly linked to a person, such as a name, identification number or location data. The data may also be indirectly linked to a person. *Sensitive data* is a special sub-category of personal data which holds extra consideration and protection in GDPR as they may give rise to strong stigmatization or discrimination in society; they include information about racial or ethnic origin, political convictions, religious or philosophical beliefs, as well as the processing of genetic and biometric data with the aim of uniquely identifying a natural person, health details or information regarding a person's sexual relationships or sexual orientation (GDPR Article 4). Secondly the *data controller* is the natural or legal person, public authority, agency or other body which determines the purposes and means of processing of personal data (GDPR Article 4 (7)), where *processing* means any operation or set of operations which is performed on personal data, such as collection, recording, organization, structuring, storage, alteration, retrieval, consultation or use (GDPR Article 4 (2)).

Article 5 of the GDPR lists the principles that apply to all personal data processing. The most important one is the PRINCIPLE OF INTEGRITY AND CONFIDENTIALITY and requires that personal data is: *processed in a way that ensures adequate personal data protection*.

A new requirement that is especially relevant for organizations using AI, is the *privacy by design* requirement. To enable privacy by design, the data controller shall build privacy protection into the system and ensure that data protection is safeguarded in the system's standard settings. This requirement is described in Article 25 of the GDPR and apply when developing software, ordering new systems, solutions and services, as well as when developing these further. The rules require that data protection is given due

---

<sup>3</sup>Colloquially, the term "artificial intelligence" is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving".

consideration in all stages of system development, in routines and in daily use. Standard settings shall be as protective of privacy as possible, and data protection features shall be embedded at the design stage. The principle of data minimization is expressly mentioned in the provision relating to privacy by design. *Data minimization* is a principle aiming to limit the amount of detail included in training or in the use of a model. This may be achieved by making it difficult to identify the individuals contained in the basic data. The use of pseudonymization or encryption techniques protect the data subject's identity and help limit the extent of intervention [6].

## 1.2 Goals and Results

This study aims to provide a methodological solution about how to design a privacy-preserving machine learning system through the usage of the Homomorphic Encryption. More precisely, this work focuses on one of the most used and versatile machine learning algorithms for image analysis, the Convolutional Neural Networks (CNNs) and presents both a methodology and a library to convert a pre-trained CNN to a privacy-preserving CNN (encoded network), by employing a Homomorphic Encryption scheme. The HE scheme considered is the one proposed by Brakerski, Fan and Vercauteren (BFV) that is mathematically based on polynomial rings. Moreover, since the usage of HE introduces some limitations in the number and types of operation performed on the encrypted data, this study focuses in understanding which is the optimal neural network to encode in the homomorphic encryption world, given a specific task to solve and some computational constraints. This is done by proposing a mathematical formulation for this problem and by solving it through the provided heuristic algorithm. In a more structured manner, this document has the following goals:

- Provide a library to convert a plain CNN in a privacy-preserving CNN end able to execute calculi on encrypted data (i.e., images).
- Propose a methodology that allows to accomplish the previously mentioned transformation of the CNN.
- Implement a heuristic methodology that allows to estimate the optimal encryption parameters (useful for the BFV HE scheme), aiming at re-

ducing the computational and memory complexities without sacrificing the classification performances.

### 1.3 Thesis Structure

Chapter 2 introduces the reader to context of the research, giving an overview on the Homomorphic Encryption, its limitations and advantages, with a strong focus on the BFV HE scheme. Moreover Chapter 2 gives an overview on the Convolutional Neural Networks and the approximations needed to allow their functioning in the HE world. Chapter 3 illustrates the current state of the art of privacy-preserving machine learning techniques, implementations and solution. Chapter 4 provides a methodology to accomplish the CNN mutation along with its mathematical formulation and gives the explanation of the heuristic used to solve the problem posed by the mathematical formulation. In Chapter 5 the methodology is revisioned taking into account also its practical implications. Chapter 6 presents the structure and implementation details of the developed Crypto CNN (CrCNN) library used to practically make a network able to predict on encrypted data. Chapter 7 shows the experimental results made to compare the proposed methodology with other techniques, to study the performances of the proposed library and heuristic. Chapter 8 reports the conclusions and the possible future works.



## Chapter 2

# Background

In this Chapter the reader will be introduced to the basics of homomorphic encryption and its limitations in Section 2.1. In Section 2.2 the Brakerski/Fan-Vercauteren homomorphic encryption scheme will be presented more in detail with its security assumptions (Subsection 2.2.3), key generations and encryption/decryption operations (Subsection 2.2.4), the possible arithmetic operations (Subsections 2.2.6 2.2.7) and the plaintext encoding (Subsection 2.2.8). A brief overview on Convolutional Neural Networks will be given in Section 2.3, along with the approximations required by the application of HE on them (Subsection 2.3.1) and the mention of some CNNs examples in Subsection 2.3.2.

### 2.1 Homomorphic Encryption

Traditional encryption schemes, both symmetric and asymmetric, were not designed to respect any algebraic structure of the plaintext and ciphertext spaces, i.e., no computations can be performed on the ciphertext in a way that would pass through the encryption to the underlying plaintext without using the secret key, and such a property would in many contexts be considered a vulnerability. An encryption scheme that allows computations to be done directly on the encrypted data is said to be a *homomorphic encryption scheme* [46]. In mathematics, a homomorphism is a *structure-preserving* transformation. For example, consider the map  $\Phi : \mathbb{Z} \rightarrow \mathbb{Z}_7$  such that  $\Phi(z) := z \pmod{7}$ . This map  $\Phi$  preserves both the additive and the multiplicative structure of the integers in the sense that for every  $z_1, z_2 \in \mathbb{Z}$  we

have that  $\Phi(z_1 + z_2) = \Phi(z_1) \oplus \Phi(z_2)$  and  $\Phi(z_1 \cdot z_2) = \Phi(z_1) \otimes \Phi(z_2)$  where  $\oplus$  and  $\otimes$  are the addition and multiplication operations in  $\mathbb{Z}_7$ . The map  $\Phi$  is a ring homomorphism (see Subsection 2.2.1) between the rings  $\mathbb{Z}$  and  $\mathbb{Z}_7$  [21]. Finding a general method for computing on encrypted data had been a goal in cryptography since it was proposed in 1978 by Rivest, Adleman and Dertouzos [58]. Since there the interest in this topic is grown due to its numerous applications.

### 2.1.1 Classification of Homomorphic Encryption Schemes

It is worth to analyze different classes of homomorphic schemes to clarify the potentiality of this type of encryption as described in [5].

Given that a circuit  $C$  is a series of computations made on some inputs, it is possible to state some preliminary definitions:

**Definition 1.** ( $\mathcal{C}$ -Evaluation Scheme). Let  $\mathcal{C}$  be a set of circuits. A  $\mathcal{C}$ -evaluation scheme for  $C$  is a tuple of probabilistic polynomial-time algorithms ( $\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec}$ ) such that:

$\text{Gen}(1^\lambda, \alpha)$  is the key generation algorithm. It takes two inputs, security parameter  $\lambda$  and auxiliary input  $\alpha$ , and outputs a key tuple  $(pk, sk, evk)$ , where  $pk$  is the public key used for encryption,  $sk$  is the secret key used for decryption and  $evk$  is the key used for the  $\text{Eval}$  algorithm.

$\text{Enc}(pk, m)$  is the encryption algorithm. As input takes the encryption key  $pk$  and a plaintext  $m$ . It outputs a ciphertext  $c$ .

$\text{Eval}(evk, C, (c_1, \dots, c_n))$  is the evaluation algorithm. It takes as input the evaluation key  $evk$ <sup>1</sup>, a circuit  $C \in \mathcal{C}$  and a tuple of inputs  $(c_1, \dots, c_n)$  that can be a mix of ciphertexts and previous evaluation results. It produces an evaluation output.

$\text{Dec}(sk, c)$  is the decryption algorithm. It takes as input the decryption key  $sk$  and either a ciphertext or an evaluation output and produces a plaintext  $m$ .

---

<sup>1</sup>The evaluation key is used to perform the relinearization operation as explained in Subsection 2.2.4

**Definition 2.** (Correct Decryption). A  $\mathcal{C}$ -evaluation scheme  $(\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$  is said to *correctly decrypt* if for all  $m$  in the plaintext space,

$$\Pr[\text{Dec}(sk, \text{Enc}(pk, m)) = m] = 1$$

This means that it must be able to decrypt a ciphertext to the correct plaintext without error.

**Definition 3.** (Correct Evaluation [15]) A  $\mathcal{C}$ -evaluation scheme  $(\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$  *correctly evaluates* all circuits in  $\mathcal{C}$  if for all inputs  $c_i$ , for each  $i = 1, \dots, n$ , for every  $C \in \mathcal{C}$ ,

$$\Pr[\text{Dec}(sk, \text{Eval}(evk, C, c_1, \dots, c_n)) = C(m_1, \dots, m_n)] = 1 - \epsilon(\lambda)$$

where  $m_i \leftarrow \text{Dec}(sk, c_i)$  and  $\epsilon$  is a negligible function.

This means that with overwhelming probability, decryption of the homomorphic evaluation of a permitted circuit yields the correct result.

A  $\mathcal{C}$ -evaluation scheme is *correct* if it satisfies the properties of correct evaluation and correct decryption. It is also *compact* if the ciphertext size does not grow too much through homomorphic operations and the output length only depends on the security parameter  $\lambda$ .

The classification of schemes is based on which kind of circuits the scheme itself can evaluate.

**Somewhat Homomorphic.** A  $\mathcal{C}$ -evaluation scheme  $(\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$  with correctness property is called *somewhat homomorphic* encryption scheme (SHE).

There are no guarantees for compactness and the set  $\mathcal{C}$  of permitted circuits does not contain all the circuits.

**Leveled Homomorphic [15].** A  $\mathcal{C}$ -evaluation scheme  $(\text{Gen}, \text{Enc}, \text{Eval}, \text{Dec})$  is called a *leveled homomorphic scheme* if it takes an auxiliary input  $\alpha = d$  to  $\text{Gen}$  which specifies the maximum depth of circuits that can be evaluated. Its properties are correctness and compactness. The latter implies that the *length of the evaluation output* (i.e., the result) does not depend on  $d$ .

The difference between SHE and leveled HE is that the depth of circuits which a SHE can handle can be increased through parameters choice - this usually means that the ciphertext size increases with the depth of the circuit allowed, while for a leveled HE the maximum depth is an input parameter and the length of the ciphertext does not depend on it.

**Fully Homomorphic Encryption** [15]. A  $\mathcal{C}$ -evaluation scheme (Gen, Enc, Eval, Dec) that is compact, correct, and where  $\mathcal{C}$  is the set of all circuits is called *fully homomorphic* encryption scheme.

This definition means that the scheme can evaluate any circuit of arbitrary size, which does not need to be known when setting the parameters.

However, it is fair to say that FHE mostly exists on papers.

### 2.1.2 Brief History of Homomorphic Encryption Schemes

RSA is the first feasible achievement of public key cryptosystem and has been introduced by Rivest, Shamir and Adleman in 1978 [59], soon after the invention of public key cryptography by Diffie and Hellman in 1976 [19]. However, the homomorphic property of RSA was shown by Rivest, Adleman and Dertouzos [58], just after the seminal work of RSA. Indeed, the first attested use of the term "privacy homomorphism" is introduced in [58]. The security of the RSA cryptosystem is based on the hardness of *factoring the product of two large prime numbers* [52]. Moreover plain RSA is not secure against *Chosen Plaintext Attacks* (CPA) which presumes that the attacker can obtain the ciphertexts for arbitrary plaintext [10], as its encryption algorithm is deterministic. RSA is only homomorphic over multiplication and this operation can be performed an unlimited number of times. Hence, it does not allow the homomorphic addition of ciphertexts. In 1982 Goldwasser and Micali (GM) [35] proposed the first probabilistic public key encryption scheme, based on the hardness of *quadratic residuosity problem* [43]. The GM scheme is homomorphic over only addition of binary numbers, however the probabilistic property of this scheme ensures *semantic security*, that is equivalent to *ciphertext indistinguishability* under CPA [36]. Intuitively, if a cryptosystem possesses the property of indistinguishability, then an adver-

sary will be unable to distinguish pairs of ciphertexts based on the message they encrypt. In 1985, Taher El-Gamal proposed a new public key encryption scheme [27] which is the improved version of the original Diffie-Hellman Key Exchange [19] algorithm and is based on the hardness of certain problems in *discrete logarithm* [49]. It is mostly used in hybrid encryption systems to encrypt the secret key of a symmetric encryption system. As RSA, El-Gamal cryptosystem is only multiplicatively homomorphic. In 1999 Paillier [54] introduced another novel probabilistic encryption scheme based on *composite residuosity* problem [42]. Paillier's encryption scheme is homomorphic over addition, but it allows also other extra basic operations.

Before 2005, all proposed cryptosystems' homomorphic properties were restricted to only either addition or multiplication operation. An important step toward an FHE scheme was the BGN SHE scheme proposed by Boneh-Goh-Nissim in [11]. BGN supports an arbitrary number of addition and one multiplication and it is able to keep constant the ciphertext size. The hardness of the scheme is based on the *subgroup decision problem* [33]. After almost 30 years from the introduction of privacy homomorphism concept [58], in his PhD thesis Craig Gentry presented the first FHE scheme that respects both addition and multiplication and proposed a general framework to obtain a FHE scheme [30]. Although Gentry's scheme was very promising it also had a lot of bottlenecks such as its computational cost in terms of applicability in real life and some of its advanced mathematical concepts made the scheme complex and hard to implement. What makes Gentry's scheme FHE are the techniques called *squashing* and *bootstrapping* that allow to evaluate unlimited number of operations. Basically squashing is a preliminary operation for the bootstrapping and aims at reducing the decryption algorithm complexity, while the bootstrapping is a "re-encrypting" procedure (see Subsection 2.2.6). Since 2009 new and more efficient schemes have been proposed such as [65] [68] [14][47] in order to address the aforementioned bottlenecks, but despite the promising theoretical power of homomorphic encryption, the practical side has remained underdeveloped for a long time.

### 2.1.3 Limitations of Homomorphic Encryption

Despite HE is a powerful instrument with many possible applications in real-world scenarios, it has some limitations that must be taken into account.

**Support for multiple users.** Traditional FHE schemes are *single-key* in the sense that they can perform (arbitrarily complex) computations on inputs encrypted under the same key. Nevertheless, Gentry [29] proposed a way of using single key FHE scheme in order to do multiparty computation, by using a joint public key and a *shared-secret key* among all parties. However, it requires an *interactive* cooperations of all parties during computations. López-Alt et al. [47] have shown promising direction to address this problem by proposing an *N-key fully homomorphic encryption scheme*. Despite these results, at the time of writing, *multi-key* FHEs are not very efficient and there are no practical implementations in any library.

**Large computational overhead** describes the ratio between the computation time in the encrypted version versus computation time in the clear. Although polynomial in size, this overhead tends to be a rather large polynomial, which increases runtimes substantially and makes homomorphic computation of complex functions impractical. One benchmark commonly used is the homomorphic evaluation of the AES [18] circuit, since it is nontrivial but also not completely infeasible. One study [31] based on the RLWE Brakerski-Gentry-Vaikuntanathan cryptosystem [13] and on its library implementation HELib [38], shows that it takes about 4 minutes and 3GB of RAM, running on a laptop Intel Core i5-3320M running at 2.6GHz, to evaluate an entire AES-128 encryption operation. With optimizations as ciphertext packing [66], called Single Instruction Multiple Data techniques, which allows several plaintexts to be encoded in a single ciphertext, it is possible to process up to 120 blocks in each such evaluation, yielding an amortized rate of just over 2 seconds per block. This implies that performances can be similar to the non-encrypted version of the AES circuit, but at the cost of increased memory consumption.

**Some inefficient operations.** For example there is no efficient way to implement the comparison operation, because it requires dealing with binary messages [51]. This latter representation brings an important issue: to perform an integer addition or multiplication with the binary representation, one must reconstruct the binary circuit of the operators.

**FHE does not necessarily imply secret function evaluation.** Indeed its goal is to allow the evaluation of arithmetic circuits on encrypted inputs, without revealing the input wire values to the evaluator. In particular, no attempt is made to keep any information hidden from the owner of the secret key [46]. For example, the homomorphic execution of the forward phase of a CNN on encrypted data does not guarantee the secrecy of the CNN model (i.e. anyone could see the structure of the network). However, it is possible to solve this problem and obtain *function privacy* in a number of ways. One way, already described by Gentry in [30], is to flood additional noise in the ciphertext. An other way is to use the GSW cryptosystem [32] as described in [12].

**Malleability.** It is a property of some cryptographic algorithms [20]. It is the ability to transform a ciphertext into a different ciphertext which will produce a new and different plaintext when decoded. That is, given an encryption of a plaintext  $m$ , it is possible to generate another ciphertext which decrypts to  $f(m)$ , for a known function  $f$ , without necessarily knowing or learning  $m$ . For example, suppose that a user sends an encrypted message to a bank containing, say, "TRANSFER \$0000100.00 TO ACCOUNT #199." If an attacker can modify the message on the wire, and can guess the format of the unencrypted message, the attacker could be able to change the amount of the transaction, or the recipient of the funds, e.g. "TRANSFER \$0100000.00 TO ACCOUNT #227". However, homomorphic encryption systems are malleable by design, allowing authorized parties to alter enciphered text without knowing the contents of the message. That is, they can manipulate an encrypted stream (set a flag, add a value) without having the ability to decipher the stream, or without expending the effort to decrypt and re-encrypt the stream. Moreover if an encrypted message is sent to an external server for outsourced computation, it must be considered a trusted party. Indeed no one can guarantee that the server has performed the desired computation. Even the RSA (see Subsection 2.1.2) is a malleable cryptosystem, however if it is used together with padding methods such as OAEP [9] this issue can be solved.

## 2.2 Brakerski/Fan-Vercauteren scheme (BFV)

One of the practical implementations of Gentry scheme [30] (see Subsection 2.1.2) is the FHE conjectured by Brakerski, Fan and Vercauteren (BFV) [26]. In this Section this homomorphic scheme will be described in detail since it is the one used for the practical part of this thesis.

### 2.2.1 Ring Definition

The BFV relies on the concept of ring, in particular on the commutative rings [25].

A commutative ring  $R$  is a set on which there are two operations defined: addition and multiplication, such that  $0 \in R$  is the identity element of addition and  $1 \in R$  is the identity element of the multiplication operation. For every element  $a \in R$  there exists an element  $-a \in R$  such that  $a + (-a) = 0$ . Furthermore, for every  $a, b, c \in R$ :

$$a(bc) = (ab)c; \quad (\text{associativity of product})$$

$$a(b + c) = ab + ac; \quad (\text{left associativity of product w.r.t addition})$$

$$(a + b)c = ac + bc; \quad (\text{right associativity of product w.r.t. addition})$$

$$a + (b + c) = (a + b) + c; \quad (\text{associativity of addition})$$

$$a + b = b + a; \quad (\text{commutative addition})$$

$$ab = ba. \quad (\text{commutative product})$$

The set of integers  $\mathbb{Z}$  is a ring as well as the set  $\mathbb{Z}_m$  of integers modulo  $m$ . If  $R$  is a given commutative ring, then the set of all polynomials in the variable  $x$  whose coefficients are in  $R$  forms the polynomial ring, denoted  $R[x]$ . In this work the focus will be on the ring  $\mathbb{Z}_m[x]$  of polynomials with integer coefficients modulo  $m$  and in particular on the polynomial ring  $\mathbb{Z}_m[x]/(x^n + 1)$ , that is polynomials of degree less than  $n$  with coefficients modulo  $m$ .

### 2.2.2 Basic Notation

In this Subsection all the mathematical entities that are employed throughout this document are exposed.

- $R = \mathbb{Z}[x]/(f(x))$  is the polynomial ring where  $f(x) \in \mathbb{Z}[x]$  is a monic irreducible polynomial of degree  $n$ . In practice a cyclotomic poly-

mial<sup>2</sup>  $\Phi_n(x)$  of degree  $n$  is used. The most popular choice for expository purposes is to take  $f(x) = x^n + 1$  with  $n = 2^d$ .

- Elements of the ring  $R$  will be denoted in lowercase bold, e.g.  $\mathbf{a} \in R$ . The coefficients of an element  $\mathbf{a} \in R$  will be denoted by  $a_i$ , i.e.  $\mathbf{a} = \sum_{i=0}^{n-1} a_i \cdot x^i$ .
- $\|\mathbf{a}\|_\infty = \max_i |a_i|$  is the infinity norm.
- Let  $q > 1$  be an integer, then  $\mathbb{Z}_q$  denotes the *set* of integers  $(-q/2, q/2]$ .
- $R_q$  is the set of polynomials in  $R$  with coefficients in  $\mathbb{Z}_q$ .
- For each  $a \in \mathbb{Z}$ , is denoted by  $[a]_q$  the unique integer in  $\mathbb{Z}_q$  such that  $[a]_q = a \pmod q$ .
- $t$  is an integer called plaintext modulo.
- The reduction to the interval  $[0, q)$  will be denoted as  $r_q(a)$  (remainder modulo  $q$ ).
- Given a probability distribution  $\mathcal{D}$ ,  $x \leftarrow \mathcal{D}$  denotes that  $x$  is sampled from  $\mathcal{D}$ , while for a set  $S$ ,  $x \leftarrow S$  denotes that  $x$  is sampled uniformly from  $S$ .
- The discrete Gaussian distribution  $D_{\mathbb{Z}, \sigma}$  over the integers is the probability distribution that assigns a probability proportional to  $\exp(-\pi|x|^2/\sigma^2)$  to each  $x \in \mathbb{Z}$ . It is used to define a distribution  $\chi$  on  $R$ .
- $\sigma$  is the standard deviation of the discrete Gaussian distribution.
- $B$  bound on the distribution  $\chi$ , that is, it is supported on  $[-B, B]$ .

### 2.2.3 RLWE Problem

Most of HE schemes base their security on the hardness of the Ring-Learning With Errors problem. The RLWE problem is a ring based version of the LWE problem. In [57] Regev gave a quantum reduction of certain approximate *Shortest Vector Problem* (SVP) [50] to LWE, i.e. if one can solve LWE, then

---

<sup>2</sup>The minimal polynomial of any  $n^{\text{th}}$  root of unity over the rationals is a cyclotomic polynomial.

there is a quantum algorithm to solve certain approximate SVP. Informally, the SVP requires a player to provide the shortest nonzero vector in a lattice, given a basis of the lattice and it is known to be an NP-hard problem. In particular, the asymptotically fastest known algorithms for obtaining an approximation to SVP on ideal lattices to within polynomial factors require time  $2^{\Omega(n)}$  [48]. Thus, the only evidence supporting the conjecture that LWE is as hard as SVP is the fact that there are no known quantum algorithms for lattice problems that outperform classical algorithms, even though this is probably one of the most important open questions in the field of quantum computing.

**Definition 4. (Decision-RLWE)** Let  $n$  be a power of 2. Let  $R = \mathbb{Z}[x]/(x^n + 1)$ , and  $R_q = \mathbb{Z}_q[x]/(x^n + 1)$  for some integer  $q$ . Let  $\mathbf{s}$  be a random element in  $R_q$  (i.e.,  $\mathbf{s} \in R_q$ ) and let  $\chi$  be the distribution on  $R_q$  obtained by choosing each coefficient of the polynomial from a discrete Gaussian distribution over  $\mathbb{Z}$ . Denote by with  $A_{\mathbf{s},\chi}$  the distribution obtained by choosing a uniformly random element  $\mathbf{a} \leftarrow R_q$  and a noise term  $\mathbf{e} \leftarrow \chi$  and outputting  $(\mathbf{a}, [\mathbf{a} \cdot \mathbf{s} + \mathbf{e}]_q)$ . The decision-RLWE problem is to distinguish  $A_{\mathbf{s},\chi}$  and the uniform distribution  $U(R_q^2)$ .

### 2.2.4 Encryption Scheme

From the decision problem reported in Subsection 2.2.3 it is possible to derive the following encryption scheme originally described in the extended version of [48].

The plaintext space is taken as  $R_t$  for some integer  $t > 1$ . Let  $\Delta = \lfloor q/t \rfloor$  and denote with  $r_t(q) = q \bmod t$ , it follows that

$$q = \Delta \cdot t + r_t(q), \tag{2.1}$$

with  $q$  and  $t$  that do not have to be prime nor coprime. The definition of the scheme mostly follows the one given in [26].

**SecretKeyGen**( $\lambda$ ) sample  $\mathbf{s} \leftarrow \chi$  and output  $sk = \mathbf{s}$

**PublicKeyGen**( $sk$ ) set  $\mathbf{s} = sk$ , sample  $\mathbf{a} \leftarrow R_q$ , small error  $\mathbf{e} \leftarrow \chi$  and output

$$pk = (\mathbf{p}_0, \mathbf{p}_1) := ([-\mathbf{a} \cdot \mathbf{s} + \mathbf{e}]_q, \mathbf{a})$$

EvalateKeyGen(sk, T) for  $i = 0, \dots, l = \lfloor \log_T(q) \rfloor$  and  $T$  is a base (independent of  $t$ ), sample  $\mathbf{a}_i \leftarrow R_q$ ,  $\mathbf{e}_i \leftarrow \chi$  and output

$$rlk = \left[ \left( [-(\mathbf{a}_i \cdot \mathbf{s} + \mathbf{e}_i) + T^i \cdot \mathbf{s}^2]_q, \mathbf{a}_i \right) : i \in [0..l] \right]$$

The *evaluation keys* are used in the relinearization phase which goal is to decrease the size of the ciphertext back to (at least) 2 after it has been increased by multiplications.

Encrypt(pk,  $\mathbf{m}$ ) to encrypt a message  $\mathbf{m} \in R_t$ , sample  $\mathbf{u}, \mathbf{e}_1, \mathbf{e}_2 \leftarrow \chi$  and output

$$ct = (\mathbf{c}_0, \mathbf{c}_1) := ([\mathbf{p}_0 \cdot \mathbf{u} + \mathbf{e}_1 + \Delta \cdot \mathbf{m}]_q, [\mathbf{p}_1 \cdot \mathbf{u} + \mathbf{e}_2]_q)$$

Decrypt(sk, ct) set  $\mathbf{s} = sk$  and  $ct = (\mathbf{c}_0, \mathbf{c}_1)$  and compute

$$\left[ \left[ \frac{t \cdot [\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q}{q} \right] \right]_t$$

The above scheme can be shown to be semantically secure assuming the hardness of RLWE [48].

### 2.2.5 Noise

All existing schemes have the common trait that they add a small “noise” component during encryption. Computing homomorphically on ciphertexts will cause these noises to grow up to the point when they become so large that decryption fails. In particular the noise growth caused by homomorphic multiplication has been the major obstacle to designing efficient schemes [26].

**Definition 5.** (Invariant Noise)[46] Let  $ct = (c_0, c_1)$  be a ciphertext encrypting the message  $m \in R_t$ . Its invariant noise  $v$  is the polynomial with the smallest infinity norm such that

$$\frac{t}{q} \mathbf{ct}(s) = \frac{t}{q} (c_0 + c_1 s) = m + v + at \tag{2.2}$$

for some polynomial  $a$  with integer coefficients.

**Lemma 6.** [46] *The function  $\text{Decrypt}(\text{sk}, \text{ct})$  as presented in Subsection 2.2.4 correctly decrypts the ciphertext  $\text{ct}$  encrypting a message  $m$ , as long as the invariant noise  $v$  satisfies  $\|v\| < 1/2$ .*

*Proof.* Let  $\text{ct} = (c_0, c_1)$ . Using the formula for decryption, for some polynomial  $A$  with integer coefficients:

$$\begin{aligned} m' &= \left[ \left[ \frac{t}{q} [c_0 + c_1 \cdot s]_q \right] \right]_t \\ &= \left[ \left[ \frac{t}{q} (c + c_1 \cdot s + Aq) \right] \right]_t \\ &= \left[ \left[ \frac{t}{q} (c + c_1 \cdot s) + At \right] \right]_t \\ &= \left[ \left[ \frac{t}{q} (c + c_1 \cdot s) \right] \right]_t. \end{aligned}$$

Then by definition of invariant noise in Eq.2.2,

$$m' = \llbracket m + v + at \rrbracket_t = m + \llbracket v \rrbracket_t.$$

Hence decryption is successful as long as  $v$  is removed by the rounding, i.e. if  $\|v\|_\infty < 1/2$ .

**Definition 7.** (Noise Budget ) [46] Let  $v$  be the invariant noise of a ciphertext  $\text{ct}$  encrypting a message  $m \in R_t$ . Then the noise budget of  $\text{ct}$  is  $-\log_2(2\|v\|_\infty)$ .

In other words, the (*Invariant*) *Noise Budget (NB)* is the left noise in the ciphertext before decryption fails.

**Lemma 8.** [46] *The function  $\text{Decrypt}(\text{sk}, \text{ct})$  as presented in Subsection 2.2.4 correctly decrypts the ciphertext  $\text{ct}$  encrypting a message  $m$ , as long as the noise budget of  $\text{ct}$  is positive.*

Even a freshly encrypted ciphertext has a non zero amount of noise as stated in the following Lemma 10.

**Lemma 9.** (*Initial Noise*) [46] *Let  $\mathbf{ct} = (c_0, c_1)$  be a fresh encryption of a message  $m \in R_t$ . The noise  $v$  in  $\mathbf{ct}$  satisfies*

$$\|v\|_\infty \leq \frac{r_t(q)}{q} \|m\|_\infty + \frac{tB}{q} (2n + 1).$$

### 2.2.6 Operations and Noise Growth

Given the interpretation of the ciphertext  $ct(s)$  as in Eq.2.2 it is possible to derive homomorphic addition ADD and multiplication MUL.

#### Addition

$$\text{ADD}(\mathbf{ct}_1, \mathbf{ct}_2) := ([\mathbf{ct}_1[0] + \mathbf{ct}_2[0]]_q, [\mathbf{ct}_1[1] + \mathbf{ct}_2[1]]_q) \text{ [26].}$$

It is possible to show that if  $\mathbf{ct}_1$  and  $\mathbf{ct}_2$  have noise  $v_1$  and  $v_2$  respectively, then the noise  $v_{add}$  in their sum is  $v_{add} = v_1 + v_2$  and satisfies  $\|v_{add}\|_\infty \leq \|v_1\|_\infty + \|v_2\|_\infty$  [46].

#### Multiplication

$\text{MUL}(\mathbf{ct}_1, \mathbf{ct}_2, \text{rlk})$  [26]: compute

$$\begin{aligned} \mathbf{c}_0 &= \left[ \left[ \frac{t \cdot (ct_1[0] \cdot ct_2[0])}{q} \right] \right]_q \\ \mathbf{c}_1 &= \left[ \left[ \frac{t \cdot (ct_1[0] \cdot ct_2[1] + ct_1[1] \cdot ct_2[0])}{q} \right] \right]_q \\ \mathbf{c}_2 &= \left[ \left[ \frac{t \cdot (ct_1[1] \cdot ct_2[1])}{q} \right] \right]_q \end{aligned}$$

The multiplication makes grow the ciphertext size, for this reason a relinearization step is necessary in order to reduce it. As showed in RELIN operation.

It is possible to prove that: if  $\mathbf{ct}_1 = (x_0, \dots, x_{j_1})$  is a ciphertext of size  $j_1 + 1$  encrypting  $m_1$  with noise  $v_1$ ,  $\mathbf{ct}_2 = (y_0, \dots, y_{j_2})$  is a ciphertext of size  $j_2 + 1$  encrypting  $m_2$  with noise  $v_2$ ,  $N_{m_1}$  and  $N_{m_2}$  are the upper bounds on the number of non-zero terms in the polynomials  $m_1$  and  $m_2$  respectively, then

the noise  $v_{mult}$  in the product  $\mathbf{ct}_{mult}$  satisfies the following bound [46]

$$\begin{aligned} \|v_{mult}\|_\infty &\leq \left[ (N_{m_1} + n)\|m_1\|_\infty + \frac{nt}{2} \cdot \frac{n^{j_1+1} - 1}{n - 1} \right] \|v_2\|_\infty \\ &\quad + \left[ (N_{m_2} + n)\|m_2\|_\infty + \frac{nt}{2} \cdot \frac{n^{j_2+1} - 1}{n - 1} \right] \|v_1\|_\infty \\ &\quad + 3n\|v_1\|_\infty\|v_2\|_\infty + \frac{t}{2q} \left( \frac{n^{j_1+j_2+1} - 1}{n - 1} \right). \end{aligned}$$

### Relinearization

RELIN( $\mathbf{ct}_{mult}$ ,  $\mathbf{rlk}$ ) [26]: write  $\mathbf{c}_2$  in base  $T$ , i.e. write  $\mathbf{c}_2 = \sum_{i=0}^l \mathbf{c}_2^{(i)} T^i$  with  $\mathbf{c}_2^{(i)} \in R_T$  and set:

$$\begin{aligned} \mathbf{c}'_0 &= \left[ \mathbf{c}_0 + \sum_{i=0}^l \mathbf{rlk}[i][0] \cdot \mathbf{c}_2^{(i)} \right]_q \\ \mathbf{c}'_1 &= \left[ \mathbf{c}_1 + \sum_{i=0}^l \mathbf{rlk}[i][1] \cdot \mathbf{c}_2^{(i)} \right]_q \end{aligned}$$

output  $\mathbf{ct}_{relin} = (\mathbf{c}'_0, \mathbf{c}'_1)$ .

It is possible to show that if  $\mathbf{ct}_{mult}$  is a ciphertext of size  $M+1$  encrypting  $m$ , and having noise  $v$ , and  $\mathbf{ct}_{relin}$  of size  $N+1$  is the ciphertext encrypting  $m$ , obtained by the relinearization of  $\mathbf{ct}_{mult}$  where  $2 \leq N+1 \leq M+1$ , then the noise  $v_{relin}$  in  $\mathbf{ct}_{relin}$  can be bounded as

$$\|v_{relin}\|_\infty \leq \|v\|_\infty + \frac{t}{q} (M - N) n B (l + 1) T \quad [46].$$

### Bootstrapping

It is a procedure that allows to turn a SHE into a FHE, since bootstrapping allows to lower the noise before to hit the maximum noise level. Gentry's idea of bootstrapping [30] consists in running the decryption  $\mathbf{SH.Decrypt}$  in the encrypted domain, i.e. homomorphically. The result of this operation produces the encryption of the same message, but with noise of a fixed size.

**Definition 10.** [5](Bootstrappable). A C-evaluation scheme is called *bootstrappable* if it is able to homomorphically evaluate its own decryption circuit plus one additional multiplication.

The bootstrapping works as follows [2]: first, it is assumed that two different public and secret key pairs are generated,  $(pk1, sk1)$  and  $(pk2, sk2)$ . The message  $m$  is encrypted  $c = \text{Encrypt}(pk1, m)$  and the secret key is encrypted too  $\text{Encrypt}(pk1, sk1)$ . Since the obtained SHE scheme (see Subsection 2.2.4) can evaluate its own decryption algorithm homomorphically, the noisy ciphertext is decrypted homomorphically using  $\text{Encrypt}(pk1, sk1)$ . Then, the result is encrypted using a different public key  $pk2$ , i.e.  $\text{Encrypt}(pk2, \text{Decrypt}(\text{Encrypt}(pk1, sk1), c)) = \text{Encrypt}(pk2, m)$ .

### 2.2.7 Operations with plaintext

There are two other important operations:

- $\text{AddPlain}(ct, m_{add})$
- $\text{MultiplyPlain}(ct, m_{mul})$

that can be seen as a particular case of the  $\text{ADD}(ct_1, ct_2)$  and  $\text{MUL}(ct_1, ct_2, r_{lk})$ . They can hugely improve performance and significantly decrease the noise budget consumption when performing addition and multiplication operations. They are useful in the case the operands are a given ciphertext  $ct$  encrypting a plaintext polynomial  $m$  and an unencrypted plaintext polynomial  $m_{add}$  or  $m_{mul}$  that does not need to be protected. The following Lemma 12 and Lemma 13 show mathematically how it is possible to perform these operations and which are the advantages in terms of growth of noise that they can bring with respect to ADD and MUL.

These operations leverage the *malleability* property that belongs by design to the homomorphic encryption schemes, as described in Subsection 2.1.3.

### Plain Multiplication

**Lemma 11.** [46](MULTIPLY PLAIN) *Let  $\mathbf{ct} = (x_0, x_1)$  be a ciphertext encrypting  $m_1$  with noise  $v$ , and let  $m_2$  be a plaintext polynomial. Let  $N_{m_2}$  be an upper bound on the number of non-zero terms in the polynomial  $m_2$ . Let  $\mathbf{ct}_{\text{pmult}}$  denote the ciphertext obtained by plain multiplication of  $\mathbf{ct}$  with  $m_2$ . Then the noise in the plain product  $\mathbf{ct}_{\text{pmult}}$  is  $v_{\text{pmult}} = m_2v$ , and can be bounded as*

$$\|v_{\text{pmult}}\|_\infty \leq N_{m_2} \|m_2\|_\infty \|v\|_\infty.$$

*Proof.* [46] By definition  $ct_{\text{pmult}} = (m_2x_0, m_2x_1)$ . Hence for some polynomials  $a, a'$  with integer coefficients,

$$\begin{aligned} \frac{t}{q} \mathbf{ct}_{\text{pmult}}(s) &= \frac{t}{q} (m_2x_0 + m_2x_1s) \\ &= m_2 \frac{t}{q} (x_0 + x_1s) \\ &= m_2 \frac{t}{q} ct(s) \\ &= m_2(m_1 + v + at) \text{ (see Eq. 2.2)} \\ &= m_1m_2 + m_2v + m_2at \\ &= [m_1m_2]_t + m_2v + (m_2a - a')t, \end{aligned}$$

where in the last line has been used  $[m_1m_2]_t = m_1m_2 + a't$ . Hence the noise is  $v_{\text{pmult}} = m_2v$  and can be bounded as

$$\|v_{\text{pmult}}\|_\infty \leq N_{m_2} \|m_2\|_\infty \|v\|_\infty.$$

### Plain Addition

**Lemma 12.** [46](ADD PLAIN) *Let  $\mathbf{ct} = (x_0, x_1)$  be a ciphertext encrypting  $m_1$  with noise  $v$ , and let  $m_2$  be a plaintext polynomial. Let  $\mathbf{ct}_{\text{padd}}$  denote the ciphertext obtained by plain addition of  $\mathbf{ct}$  with  $m_2$ . Then the noise in  $\mathbf{ct}_{\text{padd}}$  is  $v_{\text{padd}} = v - \frac{r_t(q)}{q}m_2$ , and the bound is*

$$\|v_{\text{padd}}\|_{\infty} \leq \|v\|_{\infty} + \frac{r_t(q)}{q}\|m_2\|_{\infty}.$$

*Proof.* [46] By definition of plain addition  $\mathbf{ct}_{\text{padd}} = (x_0 + \Delta m_2, x_1)$ . Hence for some polynomials  $a, a'$  with integer coefficients,

$$\begin{aligned} \frac{t}{q}\mathbf{ct}_{\text{padd}}(s) &= \frac{t}{q}(x_0 + \Delta m_2 + x_1 s) \\ &= \frac{\Delta t}{q}m_2 + \frac{t}{q}(x_0 + x_1 s) \\ &= \frac{\Delta t}{q}m_2 + \frac{t}{q}ct(s) \\ &= m_1 + v + \frac{q - r_t(q)}{q}m_2 + at \quad (\text{see Eq.2.1}) \\ &= m_1 + m_2 + v - \frac{r_t(q)}{q}m_2 + at \\ &= [m_1 + m_2]_t + v - \frac{r_t(q)}{q}m_2 + (a - a')t, \end{aligned}$$

where in the last line has been used  $[m_1 + m_2]_t = m_1 + m_2 + a't$ . Hence the noise is  $v_{\text{padd}} = v - \frac{r_t(q)}{q}m_2$  and can be bounded as

$$\|v_{\text{padd}}\|_{\infty} \leq \|v\|_{\infty} + \frac{r_t(q)}{q}\|m_2\|_{\infty}.$$

### 2.2.8 Encoding

In Subsection 2.2.4 is shown that plaintext elements in the BFV scheme are polynomials in  $R_t$  and homomorphic operations on ciphertexts are reflected in the plaintext side as corresponding operations in the ring  $R_t$ . Thus it is important to find a mapping between integers (or real numbers) and polynomials in  $R_t$  in an appropriate way in order to make applicable HE in the real world. Since no non-trivial subset of  $\mathbb{Z}$  is closed under additions and multiplications, it is needed to settle for something that does not respect an arbitrary number of homomorphic operations. It is then the responsibility of the evaluating party to be aware of the type of encoding that is used, and perform only operations such that the underlying plaintexts throughout the computation remain in the image of the encoding map [46]. In other words a number must be *encoded*.

#### Integer Encoding

Given an integer  $a$  it is required to choose a base  $S$  and to form the (up to  $n$ -bits) base- $S$  expansion of  $|a|$ , say  $(a_{n-1}x^{n-1} + \dots + a_1x + a_0)$ . When  $S = 2$ ,  $-(2^n - 1) \leq a \leq 2^n - 1$  and the expansion is a binary expansion, when  $S > 2$ , for the base- $S$  expansion, the coefficients are chosen from the symmetric set  $[-(S-1)/2, \dots, (S-1)/2]$ , since there is a unique representation with at most  $n$  coefficients for each integer in  $[-(S^n - 1)/2, (S^n - 1)/2]$ .

$$\text{IntEncode}(a, S) = \text{sign}(a) \cdot (a_{n-1}x^{n-1} + \dots + a_1x + a_0)$$

Decoding can fail for two reasons:

1. if a reduction modulo the plaintext modulus  $t$  occurs, i.e., during homomorphic operations some coefficients in the underlying encoded plaintext become greater than the plaintext modulus  $t$
2. if a reduction modulo the polynomial modulus  $x^n + 1$  occurs, i.e., during homomorphic multiplications the degree of the underlying encoded plaintext become greater than the degree of the polynomial modulus

If any of these reductions happens, decoding yields an incorrect result, but no indications of these events are given until the final decryption.

**Fractional Encoding**

Real numbers can be encoded as integers if properly scaled and rescaled back after decryption or it is possible to encode real numbers as fixed precision numbers. Just like the integer encoding (Paragraph 2.2.8 above), the fractional encoding is parametrized by an integer base  $S \geq 2$  [22]. It works by first encoding the integer part with an Integer Encoding, then the fractional part is codified in the same base- $S$ ,  $n$  is added to each exponent of the fractional part and it is converted in a polynomial by changing the base  $S$  into the variable  $x$ . Next each sign of the fractional part is flipped. For example, consider  $S = 2$  and  $r = 5.8125$  which has finite binary expansion  $5.875 = 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}$ . Its integer part is encoded as  $\text{IntEncode}(5, S = 2) = x^2 + 1$ , while its fractional part is encoded as  $\text{FractionEncode}(0.875, S = 2) = -x^{n-1} - x^{n-2} - x^{n-4}$ , thus  $\text{FractionEncode}(5.875, S = 2) = -x^{n-1} - x^{n-2} - x^{n-4} + x^2 + 1$ . More formally, for any rational number  $r$  with finite base- $S$  expansion

$$\begin{aligned} \text{FractionEncode}(r, S) = & \text{sign}(r) \cdot [\text{IntEncode}(\lfloor r \rfloor, S)] + \\ & + \text{FractionEncode}(|r| - \lfloor r \rfloor, S) \end{aligned}$$

Some rational numbers have not a finite base- $S$  expansion, thus in order to truncate them it is important to fix  $n_i$  bits for the integer part and  $n_f$  bits for the fractional part, such that  $n_i + n_f \leq n$ .

Decoding can fail for two reasons:

1. if any of the coefficients of the underlying plaintext polynomials wrap around the plaintext modulus  $t$  the result after decoding is likely to be incorrect, just as in the case of the integer encoding Paragraph 2.2.8
  
2. if during homomorphic multiplications the fractional part of the underlying plaintext polynomial expands down towards the integer part, and the integer part expands up towards the fractional part then these different parts get mixed up.

## 2.3 Convolutional Neural Networks: A Brief Overview

Convolutional Neural Networks (CNNs) is a class of deep, feed-forward artificial neural networks that are applied to analyze visual imagery. Feed-forward is the name of the classical architectures of a neural network that is a computing model whose layered structure resembles the networked structure of neurons in the brain, with layers of connected nodes. A neural network can learn from data—so it can be trained to recognize patterns, classify data, and forecast future events. The term "deep" usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150. This structure enables CNNs to extract features directly from images. The relevant features are not pre-trained; they are learned while the network trains on a collection of images. This automated feature extraction makes deep learning models highly accurate for computer vision tasks such as object classification and eliminate the need for manual feature extraction. Every hidden layer increases the complexity of the learned image features. CNNs exploit spatial locality by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that the learnt "filters" produce the strongest response to a spatially local input pattern. Stacking many such layers leads to non-linear filters that become increasingly global (i.e. responsive to a larger region of pixel space) so that the network first creates representations of small parts of the input, then from them assembles representations of larger areas. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer. The filters can start as very simple features, such as brightness and edges, and increase in complexity to features that uniquely define the object.

Employed layers can belong to one of the following groups:

- layers which role is to learn the representation and extract the learned features from the input images
- layers which role is to classify the input feature maps

Layers of the first type are modules composed by multiple convolutional and sub-sampling (pooling) layers, that, respectively, extract the features

and reduce their dimensionality and are followed by an activation function. Layers of the second type can be viewed as a classical feed-forward neural network. These modules are described more in detail in the following part of this Section.

### Convolutional Layer

From the Latin *convolvere*, “to convolve” means to roll together. For mathematical purposes, a convolution is the integral measuring how much two functions overlap as one passes over the other. One can think of a convolution as a way of mixing two functions by multiplying them. The *convolutional layer* is the core building block of a CNN. The layer’s parameters consist of a set of learnable filters (or kernels), which have a small receptive field<sup>3</sup>, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

### Pooling Layer (or Sub-Sampling)

It is common to periodically insert a *pooling layer* between successive convolutional layers in a CNN architecture. A pooling layer, has the role of summarizing a matrix of neighboring pixels into one in the resulting image. The "summarization" can be done by taking the largest value from one patch of neighbor pixels (*max pooling*), by taking the average of them (*average pooling*) or by taking their sum (*sum pooling*). The pooling function is computed on a matrix of pixels, then the subsequent matrix to which that function is applied has its center at a certain distance (called *stride*) from the previous one’s center. The intuition is that the exact location of a feature is less important than its rough location relative to other features. This layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting<sup>4</sup>. The pooling layer operates independently

---

<sup>3</sup>The area, coming from a previous layer, that goes in input to a neuron.

<sup>4</sup>The overfitting issue does not allow the CNN to generalize well when analyzing new unseen data, thus the CNN performances decrease during testing with respect to training.

on every depth slice of the input and resizes it spatially.

### Batch Normalization Layer

A *batch normalization* layer normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This layer increases the stability of the neural network, since it acts as a regularizer, it reduces overfitting, and allows to use higher learning rate during training [41].

### Activation Function

The *activation functions* basically decide whether a neuron should be activated or not as input in subsequent computations and involve a non-linear transformation of the input signal. There are different types of activation functions, as the *Rectified Linear Unit* (ReLU), the hyperbolic tangent or the sigmoid, but the ReLU is the most widely used since it has been empirically proven to be much faster and to achieve a small training error [44]. The function computed by the ReLU is  $f(x) = \max(0, x)$ .

### Fully Connected Layer

A *fully connected* layer is usually put after several convolutional and pooling layers and it is similar to a regular feed-forward neural network, since the receptive field of a neuron in a fully connected layer are all the neurons of the previous layer. The activation of a neuron in this layer can be computed with a matrix multiplication for a matrix of weights, followed by a bias offset.

### Softmax

The *softmax* layer is usually placed at the end of a Convolutional Neural Network, after the last fully-connected layer that has exactly a number of neurons equal to that of classes. The softmax function squeezes the outputs for each class between 0 and 1 and also divides by the sum of the outputs. This essentially gives the probability of the input being in a particular class. It can be defined as:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, \text{ for } j = 1, \dots, K$$

### 2.3.1 Approximations For Homomorphic Encryption

Since homomorphic encryption supports only additions and multiplications, only polynomial functions can be computed in a straightforward way. Moreover, due to the increased complexity in computing circuits with nested multiplications, it is desired to restrict the computation to low-degree polynomials [21]. Max pooling cannot be computed directly since the max-function is non-polynomial. However, powers of it can be approximated due to the relation  $\max(x_1, \dots, x_n) = \lim_{d \rightarrow \infty} (\sum_i x_i^d)^{1/d}$ . The degree  $d$  should be kept reasonably small, the smallest meaningful value  $d=1$  returns a scalar multiple of the *average pooling* function. It is also possible to compute the average pooling function by multiplying for the inverse of the input number of neurons  $n$  [22]. The sigmoid, the softmax and the ReLU activation functions are non-polynomial functions. The solution of [70] was to approximate these functions with low-degree polynomials, but another possible solution can consist in using the lowest-degree non-linear polynomial function, which is the square function  $\text{sqr}(z) := z^2$ , since it can be a good tradeoff between having a non-linear transformation, as required by the training algorithm, and the need to keep the degree of the polynomials small, to make the homomorphic encryption parameters feasible. Fortunately, the convolutional and the fully connected layer can be directly implemented, because they can be seen as weighted-sum functions that involve only additions and multiplications. Moreover during testing, the multiplications here are between precomputed weights and the additions involve precomputed biases and the values of the feeding layer. Since the weights and the biases are not encrypted, it is possible to use the more efficient plain multiplication and plain addition operations, as is described in Subsection 2.2.7.

### 2.3.2 Convolutional Neural Networks In Literature

The most recent Convolutional Neural Networks have been designed for the Image Large Scale Visual Recognition Challenge.

In 2012 the AlexNet [44] designed by Krizhevsky, Sutskever, and Hinton, won the competition. AlexNet introduced innovative techniques to solve the overfitting, as *Principal Component Analysis*<sup>5</sup> and *dropout* that consists in

---

<sup>5</sup>It is a statistical method that converts a set of possibly correlated variables in a lower dimensional set of linearly uncorrelated variables, called principal components.

ignoring randomly chosen neurons during the training phase, to reduce the possibility of learning correlations among the neurons themselves.

In 2014 the VGG [64] designed by Simonyan and Zisserman focused on empirical examination of the effects on the CNN accuracy of the depth of convolutional layers. With respect to the AlexNet, in VGG the convolutions are expanded by observing that two 3x3 convolutions are equivalent to a 5x5 one and that three 3x3 convolutions are equivalent to a 7x7 one. Furthermore, by locating a ReLU after such 3x3 convolutional layers, the CNN will learn a more discriminative function, with a smaller number of parameters.

The OverFeat [62], designed by Sermanet et al. in 2013, deals with the problem of localization useful for the classification task.

Other important CNNs are the GoogLeNet [67] designed by Szegedy et al. in 2014 and the the ResNet [40], designed by He et al. in 2015.

## Chapter 3

# State of the Art

This Chapter focuses on studies of methods for Privacy-Preserving Machine Learning and on other related interesting applications of homomorphic encryption in different fields (Section 3.3).

These works are divided in two categories that are based on one of these two scenarios:

1. There are multiple data providers whose data must be commonly protected (Section 3.1)
2. There is a single data provider (Section 3.2)

In both cases their data are sent to an outsourced server which should make some computations but which is supposed to be a *curious-but-honest* server; i.e., the server carries out the result of computations faithfully, but it may try to learn as much as possible about the inputs and/or outputs of the computation. The processed data can be sent back to the data provider/s or to a third party, a data analyst, that is allowed to decrypt the result.

### 3.1 Multiple Data Providers

A possible approach to preserve privacy while examining data on outsourced untrusted party is to use secure *Multi-Party Computation (MPC)* techniques [34], which are focused on establishing a communication protocol between the parties involved, such that if the parties follow it they will end with the desired results while protecting the security and privacy of their respective assets.

All the following works have in common the fact that they use relatively simple HE schemes, that is, additively or multiplicative HE that allow to perform homomorphically only one of these two operations. In those cases HE as RSA [58] and El-Gamal [27], that are multiplicative homomorphic or Paillier [54] that is additively homomorphic are usually used (see Subsection 2.1.2).

Barni et al. (2016) in [8] propose an iterative method to allow the privacy preserving forward phase on a neural network. The data owner encrypts the data and sends them to the server. The server computes an inner product between the data and the weights of the first layer, and sends the result to the data owner. The data owner decrypts, applies the non-linear transformation, and encrypts the result before sending it back to the server. The server can apply the second layer and send the output back to the data owner. The process continues until all the layers have been computed. However, in 2007 Orlandi et al. in [53] have shown that this process leaks information of the weights to the data owner and therefore propose a method to obscure the weights.

Kuri et al. (2017) in [45] use additively homomorphic encryption in combination with Extreme Learning Machine (ELM). ELM is a feedforward neural network where connection weights from the input layer to the hidden layer are randomly generated and the connection weights from the hidden layer to the output layer are learned analytically [16]. In particular they focus on a single hidden layer feedforward neural network (SLFN) and provide both training and testing algorithm on homomorphically encrypted data. Due to its simple structure ELM has relatively fast learning speed and higher accuracy as a nonlinear classifier and since only one homomorphic addition is performed, the computational overhead added is quite small. However, this simplicity does not allow to accomplish more complex tasks, such as object recognition in images as it is possible to do with CNNs.

Le Trieu Phong et al. (2017) in [56] propose a privacy-preserving deep learning system in which many learning participants perform neural network-based deep learning over a combined dataset of all, without actually revealing the participants' local data to a central server. They revisit the previous work in [63] and point out that local data information may be actually leaked to an honest-but-curious server, then they move on to fix that problem via building an enhanced system that also keeps accuracy intact. Each learn-

ing participant is also a data provider and has a local copy of the neural network to train. The proposed system aims at training the weights utilizing multiple data sources and gradient-encrypted Asynchronous Stochastic Gradient Descent (ASGD). There is a global weight vector  $W_{global}$ , initialized randomly. At each iteration, replicas of the neural network (one for each participant), are run over local dataset, and the corresponding local gradient vector  $G_{local}$  is sent to the server. For each  $G_{local}$ , the server then updates the global parameters  $W_{global} := W_{global} - \alpha \cdot G_{local}$  with  $\alpha$  learning rate. The updated global parameters are broadcast to all the replicas, which then use them to replace their old weight parameters, until a minimum for a pre-defined cost function is reached. As showed in Figure 3.1, to ensure privacy, additively homomorphic encryption is used and the updating formula becomes  $E(W_{global}) := E(W_{global}) + E(-\alpha \cdot G_{local})$  and to ensure integrity of the homomorphic ciphertext, each client uses a secure channel such as TLS/SSL to communicate the homomorphic ciphertexts to the server. To use the power of parallel computation when the server has multiple processing units  $PU_1, \dots, PU_{n_{pu}}$ , ASGD splits the weight vector  $W$  and gradient vector  $G$  into  $n_{pu}$  parts, so that each processing unit  $PU_i$  computes the update rule  $W_i = W_i - G_i$ .

The participants jointly set up the public key  $pk$  and secret key  $sk$  for an additively homomorphic encryption scheme. The secret key  $sk$  is kept confidential against the cloud server, but is known to all learning participants. Each participant establishes a TLS/SSL secure channel, different from each other, to communicate and protect the integrity of the homomorphic ciphertexts. The downloads and uploads of the encrypted parts of  $W_{global}$  can be asynchronous in two aspects: the participants are independent with each other; and the processing units are also independent with each other. However protecting the gradients against the cloud server comes with the cost of increased communication between the learning participants and the cloud server. Saeed Samet et al. (2012) in [61] focus their work on training ELM networks when data is vertically or horizontally distributed among parties (data owners), using as sub-protocols secure multi-party multiplication and secure multi-party addition. The model (i.e. the network) is securely constructed and distributed among the parties involved. Indeed, at the end of the learning protocol the parties can jointly use the model on target data to predict the corresponding output. This seems to be a different approach with

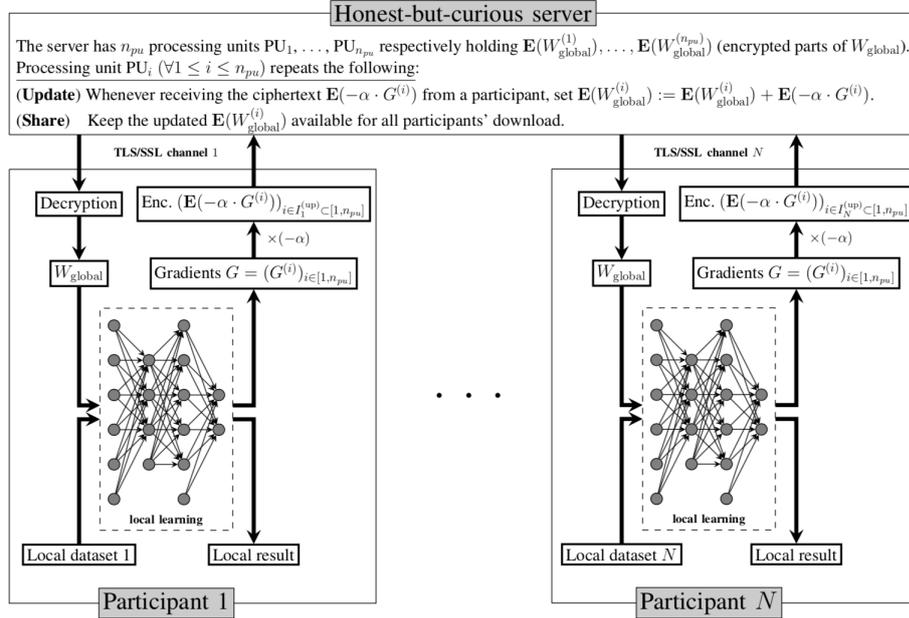


Figure 3.1: Gradients-encrypted Asynchronous SGD for privacy-preserving deep learning, with a curious cloud server and  $N$  honest participants. [56]

respect to the the two previously presented works ([45], [56]) where there was always a central entity (server/cloud) designated to aggregate partial results coming form the different parties. They also prove that their secure multi-party multiplication as well as secure multi-party addition are secure against collusion attacks<sup>1</sup> [60] (if up to  $n - 1$  parties are implicated in the attack) and can be used over public channels. The problem with this protocol is that the time needed for training increases with the number of parties involved.

### 3.2 Single Data Provider

Graepel et al. (2013) [37] suggest a way to adapt machine learning algorithms, as classification, in order to train them over encrypted data. Therefore, for homomorphic encryption limitations, they are forced to use functions that learn low degree polynomials. As a result, most of the algorithms proposed are of the linear discrimination type and therefore for many tasks they do not deliver the same level of accuracy as neural networks are capable

<sup>1</sup>The operations that combine several data to produce a new copy. The operations include averaging, replacement, linear combination, etc.

of.

Yoshinori Aono et al. (2016) in [4] have proposed a homomorphism-aware logistic regression via approximation of the logarithm in the cost function to minimize, with a polynomial of degree  $k$  (e.g.  $k=2$ ). Since HE ensures data secrecy, *differential privacy* is added to the model to ensure also output privacy.

*Differential privacy* is a technique that aims to maximize the accuracy of queries from statistical databases<sup>2</sup> while measuring impact on individuals whose information is in the database. Intuitively it can be explained by saying that if there are two datasets  $D_1$  and  $D_2$  that differ on a single element (i.e., the data of one person), a given differentially private algorithm will behave approximately the same on both datasets. More formally,

**Definition 13.** ( $\epsilon$ -Differential Privacy [24]) A randomized mechanism  $M$ , that answers to queries on a database, provides  $\epsilon$ -differential privacy, if, for all databases  $D_1$  and  $D_2$  which differ by at most one element, and for any  $t$ ,

$$\frac{Pr[M(D_1) = t]}{Pr[M(D_2) = t]} \leq e^\epsilon.$$

It has been shown in [23] that if a mechanism satisfies  $\epsilon$ -differential privacy, then an adversary who knows the private value of all the individuals in the dataset, except for one single individual, cannot figure out the private value of the unknown individual, with sufficient confidence, from the responses of the mechanism  $M$ .  $\epsilon$ -differential privacy is therefore a very strong notion of privacy [17].

In [4] to add  $\epsilon$ -differential privacy they change the computation done at the server side. Namely, the server generates Laplace noises, encrypts and sums them to its computation. The result is that the data analyst (or data provider), finds, after decryption, exactly the Laplace-perturbed coefficients of the cost function. Even if good results are obtained in the experimental result in terms of F-score and AUC (Area Under The Curve) measure, differential privacy in addition to the encryption decreases the performances of the classifier, this imply that the usage of really large datasets is needed in

---

<sup>2</sup>Here, the term statistical database means a set of data that are collected under the pledge of confidentiality for the purpose of producing statistics that, by their production, do not compromise the privacy of those individuals who provided the data

order to mitigate the effect of noise introduced by differential privacy. One of the most important works that is considered to be the first and unique example of application of HE to CNNs is the one presented by Dowlin et al. (2016) [21]. The difference with respect all the other works is given by the fact that CNNs are really powerful classification machine learning model and are way more complex than ELM or logistic regression. They propose a method to convert pre-trained neural networks to CryptoNets, neural networks that can be applied to encrypted data. This work focuses only on the inference stage and the assumption is that the cloud (that has to perform the prediction) already has the model. CryptoNets are able to make predictions on encrypted data coming from a data provider on an outsourced server that provides back to the data owner the results of computation, still in an encrypted form. The data owner then is able to decrypt the results and access to the unencrypted prediction. In this sense the purpose of this work is similar to the one of this thesis, however they do not provide an open source library for CNNs that works homomorphically on encrypted data that can reproduce their work and they do not focus on the estimate of optimal parameters for the HE conversion of the CNN, even if this is an important starting point to achieve better performances. Another difference is the fact that they apply batching techniques in order to pack more input data in the same ciphertext, using the Chinese Remainder Theorem (CRT) [25] to perform Single Instruction Multiple Data (SIMD) operations [31]. This gives good throughput but relatively poor latency in terms of predictions per hour.

### 3.3 Other related works

One of the first attempt made to speed up the computation proposed in CryptoNets [21], as presented in the previous Section 3.2, has been made by Yizhi Wang et al. (2018) in [69]. In particular they propose an hardware solution, by providing a dedicated convolution core architecture for the most complex convolutional layers of CryptoNets on the basis of Fan-Vercauteren (FV) HE scheme [26]. They propose a simplified modular multiplication algorithm suitable for CryptoNets that enable to simplify and reduce dramatically hardware complexity. Compared to a well optimized version CPU implementation of CryptoNets, their proposed architecture is  $11.9\times$  faster while consuming a power of  $537mW$ .

Zhenyong Zhang et al. (2018) in [72] propose the application of HE for secure Kalman Filter state estimation in cyber-physical systems. In their scenario, the data providers are multiple sensors that send their measurements to an outsourced estimator node in charge of computing the Kalman Filter prediction. The goal of using HE is to protect data against confidentiality attacks in the communication network or at the estimator node. They consider that sensor measurements are valid but they can be overheard during communications. Furthermore, they assume a curious-but-honest estimator; i.e., the estimator carries out the estimation faithfully, but it may try to learn as much as possible about the inputs and/or outputs of the estimation. They use RSA since there is only the need of a multiplicative homomorphic encryption.



## Chapter 4

# Parameters Estimation Methodology

In this Chapter the encryption parameters required by the BFV HE scheme (see Section 2.2) will be presented in a more detailed way and the relations that exist among them will be analyzed (Section 4.1). The problem of transposing a plain CNN in the HE world and thus of finding appropriate encryption parameters, that this work aims to solve, will be exposed in Section 4.2, and in Section 4.3 the problem will be described in a more formal manner, through a mathematical optimization problem. The solution proposed involves the implementation of a heuristic and will be explained in Section 4.4.

### 4.1 Encryption Parameters

The choice of encryption parameters is crucial since it significantly affects the performance, capabilities, and security of the encryption scheme. Some choices of parameters may be insecure, give poor performance, yield ciphertexts that will not work with any homomorphic operations, or a combination of all of these. In Table 4.1 there is a list of the most important parameters of the BFV encryption scheme, that has been explained in Section 2.2 and below it is explained which are the relations that these parameters have each other.

Before to encrypt a number it must be encoded as a plaintext polynomial as explained in Subsection 2.2.8. Indeed in BFV the plaintext space is

Table 4.1: Notation used

Parameter	Description
$q$	Modulus in the ciphertext space
$t$	Modulus in the plaintext space
$n$	A power of 2

$R_t = \mathbb{Z}_t[x]/(x^n + 1)$ , that is, polynomials of degree less than  $n$  with coefficients modulo  $t$ . Thus, it is important that the coefficients of the polynomials appearing throughout the computations never experience coefficients larger than the plaintext modulus  $t$ , otherwise the decryption procedure will lead to incorrect results.

The **plaintext modulus**  $t$  in the BFV scheme can be any positive integer. There is not a prescribed plaintext polynomial that is better than others, because it depends on the calculi (i.e. in the considered case, on the structure of the CNN) and data that is needed to perform. Nevertheless, it is not easy to understand which is the maximum infinity norm of a polynomial that is reached throughout the computations, but it is a key point not to lose integrity of the expected result. The plaintext modulus influences also the so called Noise budget NB, that has been presented in Subsection 2.2.5. As explained before, the NB determines the number of operations one can perform on a ciphertext (multiplications, additions, exponentiations, both with another ciphertext or plaintext) without compromising the final result. More precisely, the value of the NB in a freshly encrypted ciphertext, given in Definition 7 can be approximated by the following formula:

$$NB \sim \log_2(\text{coeff\_modulus}(q)/\text{plain\_modulus}(t)) \text{ (bits)} \quad (4.1)$$

The polynomial  $(x^n + 1)$  is the **polynomial modulus**, where  $n$  is a power of 2. Indeed as explained in Section 2.2, the discrete Gaussian distribution  $D_{\mathbb{Z},\sigma}$  is used to define a distribution  $\chi$  on  $R$ . The distribution  $\chi$  is in general not as simple as just sampling coefficients according to  $D_{\mathbb{Z},\sigma}$ . However, for the polynomial  $(x^n + 1)$  where  $n$  is a power of 2,  $\chi$  can indeed be defined as  $D_{\mathbb{Z},\sigma}^n$  [26], from which is possible to sample in a more efficient way. Increasing  $n$  significantly decreases performances since operations are performed over

Table 4.2: Default pairs  $(n, q)$  for 128-bit, 192-bit, and 256-bit  $\lambda$ -security levels.

$n$	Bit-length of $q$		
	128-bit security	192-bit security	256-bit security
1024	27	19	14
2048	54	37	29
4096	109	75	58
8192	218	152	118
16384	438	300	237
32768	881	600	476

polynomials of bigger degree.

If the polynomial modulus is held fixed, then the choice of the **coefficient modulus**  $q$  affects: the noise budget in a freshly encrypted ciphertext (see Eq.(4.1)) and the security level (see Table 4.2). In general it is possible to take as  $q$  any integer as long as it is not too large to cause security problems. In Table 4.2 are showed the upper bounds on the bit length of  $q$  with respect to a given security level  $\lambda$  and a polynomial modulus degree  $n$ , assuming the standard deviation of the discrete Gaussian distribution  $\sigma$  to be the default value of  $3.19 \approx 8/2\pi$  [26]. These values are retrieved according to the most recent homomorphic encryption security standards. Table 4.2 shows that using a larger  $n$  allows for a larger  $q$  to be used without decreasing the security level, which in turn increases the noise ceiling and thus allows for larger  $t$  to be used. For this reason, decreasing  $q$  with all the other parameters held fixed, only increases the security level. On the other hand, given a certain  $q$ , choosing a big  $t$  allows to reach bigger infinity norm for polynomials during the computation, but at the same time decreases the initial NB and thus the possibility to evaluate a longer circuit (number of operations) on input data. Indeed, each operation performed adds a certain amount of noise to the initial ciphertext and the NB in a freshly encrypted number is the max amount of noise is it possible to add, as detailed in Subsection 2.2.5. Note that the NB computed is a pessimistic estimate and not a precise computed value and that the NB consumed depends not only on the type of operation performed (multiplication rather than addition), but also on the encryption parameters  $q$  and  $t$  chosen.

## 4.2 Problem Formulation

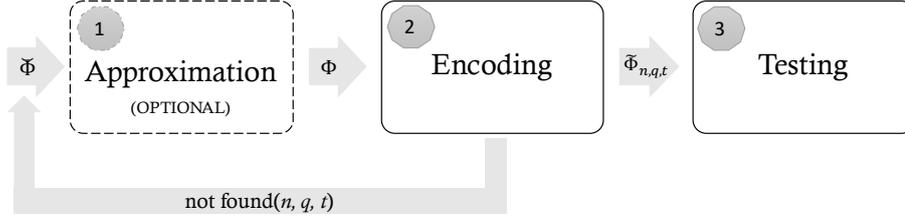
The task of preserving privacy of data provider while its data are processed by an outsourced CNN on a server can be achieved in different ways. Given that HE is used to guarantee the privacy of data, there can be two possibilities:

1. One can train the CNN from scratch on plain publicly available datasets or take a publicly available pre-trained CNN. In the latter case the CNN must be transformed in a way that allows to compute only polynomial functions, to be compliant with the operations that it will be able to perform on encrypted data (as explained in Subsection 2.3.1). Then one can encode the CNN and send it to the server along with the parameters  $n$ ,  $t$  and  $q$  used for the encoding. The encoded network can then be used to make prediction on encrypted data, provided that they are encrypted with the same encryption parameters.
2. The data provider can encrypt its data and send them to the server. The server trains a CNN directly on encrypted data.

The second option could look simpler than the first one, but it would require to run the backward propagation algorithm homomorphically on the encrypted data, that is a really computational intensive process. Moreover, since only polynomial functions could be used and thus functions like rectified linear and sigmoid functions should be omitted, some derivatives could become unbounded. This could lead to strange behavior when running the gradient descent algorithm during the backward propagation step, especially for deeper nets it would sometimes blow up or overfit.

The first option becomes the more promising way to follow and basically require the steps in Figure 4.1. At each step the accuracy of the original CNN can decrease by a factor  $\epsilon \geq 0$ , since it is gradually changed and HE parameters are introduced, but it becomes able to guarantee the privacy of the processed data.

**STEP 1 APPROXIMATION (OPTIONAL):** If the CNN  $\check{\Phi}$  in input contains non-polynomial functions, it is *approximated* with only polynomial functions as explained in Subsection 2.3.1 and the approximated CNN  $\Phi$  is eventually retrained.

Figure 4.1: Transformation steps of the pre-trained CNN  $\check{\Phi}$ .


**STEP 2 ENCODING:** The approximated  $\check{\Phi}$  is *encoded* as explained in Sub-section 2.2.8 provided that optimal encryption parameters  $n$ ,  $t$  and  $q$  are found, otherwise one should return to STEP 1 to find a smaller and different CNN  $\check{\Phi}$ . This step outputs  $\check{\Phi}_{n,q,t}$  that is an encoded network under parameters  $n$ ,  $t$  and  $q$ . In Section 4.3 this step will be explained in detail.

**STEP 3 TESTING:** The encoded  $\check{\Phi}_{n,q,t}$  is ready to be *tested* and thus make predictions on encrypted data.

### 4.3 Optimization Problem

Section 4.1 gives a hint of how difficult it is, in general, to find valid encryption parameters. This becomes even more difficult if the field of application of the HE is the one of Deep Learning and CNNs where there are many nested computations to perform (i.e., the depth of the circuit to compute on input data is generally high) and the number of parameters (i.e., weights) that composes the CNN's model is huge as shown in [3].

If  $I$  is a  $r \times c \times d$  input image of  $r$  rows,  $c$  columns and  $d$  channels (e.g.  $d=3$  for RGB images and  $d=1$  for grey-scale images),  $y \in \Psi = \{w_1, \dots, w_\Psi\}$  is the output of the CNN representing the multi-class classification of image  $I$  and  $\check{\Phi}$  is the approximated pre-trained model of a CNN, defined as  $y = \check{\Phi}(I)$ , there are two objectives that can be defined and must be pursued:

1. Transform the approximated CNN's pre-trained model  $\Phi$  in  $\tilde{\Phi}$  (*encoding*), such that the transformed  $\tilde{\Phi}$  is able to accomplish the same classification task of the original  $\Phi$  but on encrypted input data  $Enc(I)$ .
  
2. Find suitable encryption parameters  $n$ ,  $t$ , and  $q$  that allow to perform this transformation in an optimal way.

Since homomorphic encryption supports only additions and multiplications (see Subsection 2.3.1), only polynomial functions can be computed in a straightforward way, thus the transformation of  $\Phi$  in  $\tilde{\Phi}$  can be achieved if  $\Phi$  contains only polynomial functions. The transformation of  $\Phi$  in  $\tilde{\Phi}$  consists basically in an encoding of all the parameters  $\theta$  of  $\Phi$ , i.e., each  $\theta$  becomes  $\tilde{\theta}$  that is, a polynomial in the plaintext space  $R_t$ . The encoding methodology has been explained in Subsection 2.2.8. For this reason the model  $\tilde{\Phi}$  is not encrypted, but only encoded, and most of the operations can be performed, in a more efficient way, as PLAIN MULTIPLICATIONS and PLAIN ADDITIONS (see Subsection 2.2.6).

More formally, the previous two objectives can be seen as the output of a function  $f$  that, given the CNN's model  $\Phi$ , the plain input dataset  $D$  such that  $I \in D$  with its corresponding classifications  $Y$  given by the model, the set of allowed coefficient modulus  $Q_{set}$ , the upper bound for the degree  $n$  of the polynomial modulus  $N_{max}$ , the range of values in which the plaintext modulus  $t$  can vary  $T_{range}$  and the desirable level of security  $\lambda$ ; provides the CNN's model  $\tilde{\Phi}_{n,q,t}$  encoded with the optimal  $n$ ,  $q$  and  $t$  :

$$\tilde{\Phi}_{n,q,t} = f(\Phi, D, Y, N_{max}, Q_{set}, T_{range}, \lambda)$$

Figure 4.2 shows an approximated CNN  $\Phi$  with  $l$  layers, where  $\phi_{\theta_i}^{(i)}$  with  $i = 1, \dots, l$  is the function with parameters  $\theta_i$  computed in the  $i$ -th layer, that receives as input the computation of the previous layer  $i - 1$ . Below  $\Phi$  there is  $\tilde{\Phi}_{n,q,t}$ , that is the CNN transformed by the function  $f$  as explained before.

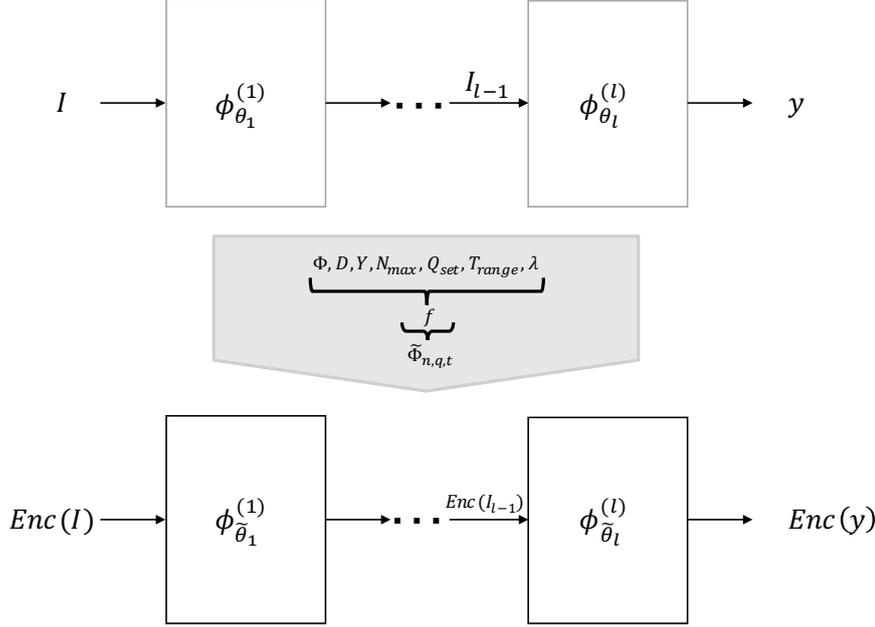


Figure 4.2: General structure of a CNN  $\Phi$  and below its transformation in  $\tilde{\Phi}_{n,q,t}$  in the encryption world accomplished by function  $f$ .

The function  $f$  can be modeled by the Optimization Problem 1. The first term of the objective function is the prediction error of the encoded CNN  $\tilde{\Phi}$ , that can cause a significant loss in accuracy with respect to the one of the original model  $\Phi$ . There is also the need to find the minimal  $n$ ,  $q$  and  $t$  that preserve the original accuracy of the model in order to introduce the smallest possible performance's overhead in terms of memory and computational complexities required by the encoded model  $\tilde{\Phi}$ . One can set parameters  $\alpha, \gamma, \delta, \beta$  depending on its needs and the application field of the CNN. For example  $\alpha$  can be set to be small if a bigger prediction error is tolerated.

These parameters are subject to a number of constraints given by the BFV HE scheme that has been previously described in Section 2.2.

In detail, as mentioned before in Section 4.1 the plaintext modulus  $t$  must be smaller than the coefficient modulus  $q$  (Eq. (4.2)) and  $q$  must be at least two (Eq. (4.3)). Constraint in Eq.(4.4) is of crucial importance as written in Section 4.1 and requires that the infinity norm of every generic computation in  $\tilde{\Phi}$ , both internal to a single layer or at the end of it, and thus the encrypted polynomial that represent this result, never exceed the plaintext modulus  $t$ . Constraint in Eq.(4.5) refers to the one introduced in Eq.(4.1)

and states that the NB in the freshly encrypted image must be sufficiently large to support all the operations performed by each layer in  $\tilde{\Phi}$ , thus the NB consumed by each computation that depends itself by the encryption parameters selected. Constraint in Eq.(4.6) is referred to the security of the encryption since the total bit count of  $q$  must be smaller than the upper bound (UB) showed in Table 4.2 for a given security level  $\lambda$  and polynomial modulus degree  $n$ . The last constraint (4.7) states that  $n$  must be a power of 2.

Optimization problem 1

$$\min_{n,q,t} \alpha \sum_i^D (y_i - \tilde{y}_i)^2 + \gamma t + \delta n + \beta q$$

subject to

$$t < q \tag{4.2}$$

$$q \geq 2 \tag{4.3}$$

$$t > \max \|p\|_\infty \text{ with } p \in R_t \text{ and generic intermediate result of } \tilde{\Phi} \tag{4.4}$$

$$\log_2 \left( \frac{q}{t} \right) > \sum_{i=1}^l NB_{n,q,t}(\phi_{\theta_i}^{(i)}) \tag{4.5}$$

$$\log_2 q \leq UB(\lambda, n) \tag{4.6}$$

$$n = 2^d \text{ s.t. } d \in N \tag{4.7}$$

$$t, q, n \in \mathbb{N}$$

$$\alpha, \beta, \gamma, \delta \geq 0$$

The real hard problem to solve is the exact calculus of the right term of constraint in Eq. (4.4) that is valid for every possible input data. Indeed each operation in the model  $\tilde{\Phi}$  on an input polynomial  $p$  causes a grow of the initial infinity norm of  $p$ , but since operations are performed homomorphically (i.e., the polynomial in input is encoded first as a polynomial in  $R_t$ , as described in Subsection (2.2.8), and then is encrypted as a polynomial in  $R_q$  (see Subsection 2.2.4)), only after decryption is achievable to understand if a reduction modulo  $t$  has occurred, since the result is different from the expected one. One can observe that it is possible to estimate the grow of such infinity norm by considering the specific operation to perform (e.g. addition)

and considering the shape of the two polynomials in input, before to encrypt them. However, this approach can be useful only if a very small number of operations should be evaluated on the input ciphertext and this is not the case of CNNs, because otherwise this estimate would be so rough that no valid plaintext modulus  $t$  would be found, as will be informally explained in Subsection 7.4.4. Moreover, if it is not possible to find a valid  $t$ , then it is impossible to satisfy the constraint in Eq. (4.5).

This leads to the impossibility of writing in a closed form the constraints in (4.4) and (4.5) and suggests the need to find an alternative solution for the problem that function  $f$  should solve.

## 4.4 Heuristic: Binary Search Of Plaintext Modulus

The idea is to implement a heuristic method able to find a suboptimal plaintext modulus  $\tilde{t}$ , that approximates the optimal  $t$  with a certain error  $\epsilon$ .

The heuristic is a sort of binary search that gets as inputs: the model  $\Phi$ , a number of  $K \subseteq |D|$  images to test, where  $D$  is the entire test set, an ordered list  $t_{list}$  of possible plaintext moduli  $t$  with two indexes pointing to the minimum and maximum value of that list, a security parameter  $\lambda$  and a tentative starting big coefficient modulus  $q_0$ . It recursively tests the number in the middle of the two boundaries as  $t$ . By testing, it is intended the execution on the CNN's model  $\tilde{\Phi}_{n,q,t}$  of the forward phase on  $K$  images sampled uniformly from the dataset  $D$  as showed by the pseudocode 4.2. If the testing for a given  $t$  is successful, that is  $\tilde{y}_i = y_i$  for each  $i = 0, \dots, K$ , where  $\tilde{y}_i$  is the decrypted prediction given by the encoded CNN  $\tilde{\Phi}_{n,q,t}$  for image  $i$  (i.e.,  $\tilde{y}_i = Dec(\tilde{\Phi}_{n,t,q}(Enc(I_i)))$ ) and  $y_i$  is the prediction of the plain CNN  $\Phi$  for the same input image  $i$  or if the starting NB in the ciphertext is not sufficient to carry out the entire computation until the final layer, the search continues on the left interval, to find if it exists, a smaller plaintext modulus. This is done because if the testing is successful the heuristic wants to find the smallest possible  $t$ , while if the noise budget is insufficient, by decreasing  $t$  the noise budget is increased (see Eq.4.1). Instead, if exists an image  $i$  such that  $\tilde{y}_i \neq y_i$ , i. e., there is a misprediction with respect to the original CNN  $\Phi$ , the search continues on the right interval. Note that if the search is performed only on the powers of two, i.e.  $t_{list} = [2, 2^2, 2^3 \dots, 2^{59}]$ ,

for the formula of the NB Eq.(4.1), the initial amount of NB changes significantly between different runs of the binary search, while if the search is carried on all integers i.e.,  $t_{list} = [2, 3, 4, \dots, 2^{59}]$  the difference in NB between successive runs of the algorithm is not really significant. Algorithm 4.1 shows the pseudocode of the described binary search, while Algorithm 4.2 describes in detail how the testing is performed.

The Algorithm 4.1 is able to find a parameter  $\tilde{t}$  for every given CNN in input if a subdivision as in Figure 4.3 (a) is possible for the numbers in the given range. The found  $\tilde{t}$  lies in the SUCCESS interval and tends to be as close as possible to the optimal  $t$ . The case of Figure 4.3 (b) instead, shows the worst scenario where is not found any valid  $\tilde{t}$ . The only possible solution is to increase the stating  $q_0$ , so to reduce the OUT OF BUDGET INTERVAL and run again the binary search, to bring back to case (a).

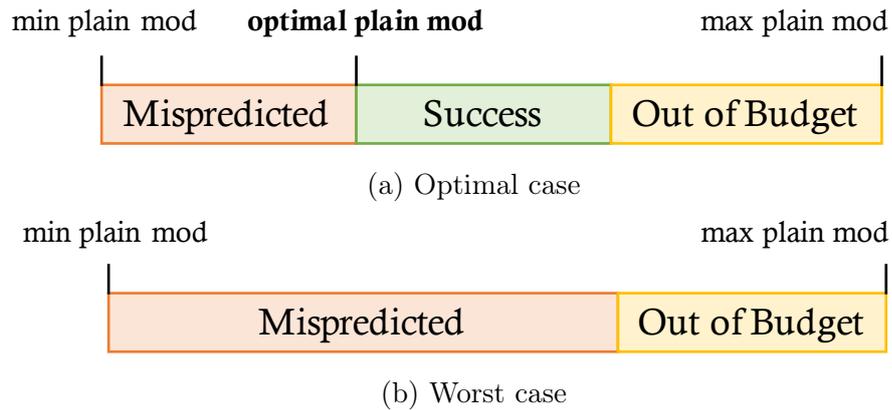


Figure 4.3: Binary Search Cases

---

**Algorithm 4.1** Plaintext Modulus Binary Search

---

**Input:** Model  $\Phi$ , sorted list  $t_{list}$  of possible plain modulus  $t$ , five integers  $t_{min}$  and  $t_{max}$  s.t.  $T_{range} = [t_{list}[t_{min}], t_{list}[t_{max}]]$ , security level  $\lambda$ , given upper bound for the coefficient modulus  $q_0$ , number of  $K$  images to test

**Output:** Plain modulus  $\tilde{t}$  or NOT\_FOUND\_T

```

1: function SEARCH_T( $\Phi, t_{list}, t_{min}, t_{max}, \lambda, q_0, K$ )
2:   if  $t_{max} - t_{min} = 1$  then                                     ▷ Base Case
3:      $test \leftarrow$  TEST_PLAIN_MOD( $\Phi, t_{list}[t_{min}], \lambda, q_0, K$ )
4:     if  $test = \text{SUCCESS}$  then
5:       return  $t_{list}[t_{min}]$ 
6:     end if
7:     if  $test = \text{OUT\_OF\_BUDGET}$  then
8:       return NOT_FOUND_T
9:     end if
10:    if TEST_PLAIN_MOD( $\Phi, t_{list}[t_{max}], \lambda, q_0, K$ ) = SUCCESS then
11:      return  $t_{list}[t_{max}]$ 
12:    end if
13:    return NOT_FOUND_T
14:  end if
15:   $t_{index} \leftarrow t_{min} + (t_{max} - t_{min})/2$ 
16:   $test \leftarrow$  TEST_PLAIN_MOD( $\Phi, t_{list}[t_{index}], \lambda, q_0, K$ )
17:  if  $test = \text{SUCCESS}$  or  $test = \text{OUT\_OF\_BUDGET}$  then   ▷ Go Left
18:     $t_{smaller} \leftarrow$  SEARCH_T( $\Phi, t_{list}, t_{min}, t_{max} - 1, \lambda, q_0, K$ )
19:    if  $t_{smaller} > 0$  then                                     ▷ smaller p_mod found
20:      return  $t_{smaller}$ 
21:    end if
22:    if  $test = \text{SUCCESS}$  then   ▷ smaller p_mod not found, prev ok
23:      return  $t_{list}[t_{index}]$ 
24:    end if
25:    return NOT_FOUND_T ▷ smaller p_mod not found, prev not
    ok
26:  end if
27:  if  $t_{index} \geq t_{max}$  then                                     ▷ p_mod needed  $\notin T_{range}$ 
28:    return NOT_FOUND_T
29:  end if
30:  return SEARCH_T( $\Phi, t_{list}, t_{min} + 1, t_{max}, \lambda, q_0, K$ )   ▷ Go Right
31: end function

```

---

---

**Algorithm 4.2** Test plaintext modulus

---

```

1: global variables
2:    $Y$  ▷ predictions given by model  $\Phi$ 
3:    $D$  ▷ dataset
4: end global variables
Input: Model  $\Phi$ , four integers plain modulus to test  $t_{test}$ , security level  $\lambda$ ,
        upper bound for the coefficient modulus  $q_0$ , number of  $K$  images to test
Output: SUCCESS, OUT_OF_BUDGET or MISPREDICTED
5: function TEST_PLAIN_MOD( $\Phi, t_{test}, \lambda, q_0, K$ )
6:    $t \leftarrow t_{test}$  ▷ set encryption parameters
7:    $q \leftarrow q_0$ 
8:    $n \leftarrow \lambda(q_0)$ 
9:    $sk \leftarrow \text{GEN\_SEC\_KEY}(n, q, t)$ 
10:   $pk \leftarrow \text{GEN\_PUB\_KEY}(sk)$ 
11:   $\tilde{\Phi} \leftarrow \text{ENCODE}(\Phi, n, t)$  ▷ Transform each  $\theta_i$  in  $\tilde{\theta}_i$ 
12:  for  $i \leftarrow 0$  to  $K$  do
13:     $I \leftarrow \text{RANDOMLY\_SAMPLE\_AT\_UNIFORM}(D)$ 
14:     $Enc(I) \leftarrow \text{ENCRYPT}(\text{ENCODE}(I, n, t), pk)$ 
15:    try
16:       $Enc(y_i) \leftarrow \tilde{\Phi}(Enc(I))$  ▷ Forward
17:    catch Out_Of_Budget_Exception
18:      return OUT_OF_BUDGET
19:    end try
20:     $\tilde{y}_i \leftarrow \text{DECRYPT}(Enc(y_i), sk)$ 
21:    if  $\tilde{y}_i \neq y_i$  then
22:      return MISPREDICTED
23:    end if
24:  end for
25:  return SUCCESS
26: end function

```

---

## Chapter 5

# Practical Considerations About The Proposed Methodology

In Section 4.1 the encryption parameters with their relationships have been described from a theoretical point of view depending on the underlying BFV scheme that has been considered.

In Section 5.2 of this Chapter these parameters will be considered with respect to the practical implementation of the BFV scheme given by the Simple Encryption Arithmetic Library (SEAL 2.3.1) (Section 5.1). Moreover, in Section 5.3, the parameters estimation problem presented in Section 4.3 will be revisited according to the other constraints imposed by the practical application of the BFV HE scheme.

### 5.1 Simple Encrypted Arithmetic Library (SEAL 2.3.1)

The Simple Encrypted Arithmetic Library is written in the C++ language and implements the BFV encryption scheme by offering primitives to handle operations, encryption and decryption procedures. However, the library differs in some aspects from the original BFV scheme. Indeed the bootstrapping operation (see Subsection 2.2.6) is not implemented. For this reason, the number of operations that is possible to perform on a ciphertext is not unlimited as in a Fully HE scheme, but it depends on the encryption parameters chosen as in a Somewhat HE scheme. Moreover, the compactness property (see Subsection 2.1.1) is not preserved in SEAL, since ciphertexts

can be of arbitrary length (at least two) and not of fixed length as in the original BFV scheme. SIMD techniques to perform batching and operations ciphertext-plaintext (see Subsection 2.2.7) are allowed. Finally there are some constraints on the encryption parameters as described in Section 5.2.

## 5.2 Practical Encryption Parameters

The **plaintext modulus**  $t$  in SEAL can be any positive integer of size between 2 and 60 bits, thus  $2 \leq t \leq 2^{59}$ .

In the **polynomial modulus**  $(x^n + 1)$ ,  $n$  is a power of 2 as described before. For performance reasons in SEAL the **coefficient modulus**  $q$  is a product of multiple small primes  $q_1 \times \dots \times q_k$ , even if taking these  $q_k$  to be of special form does not provide any additional performance improvement. Therefore, it is possible to choose a set of arbitrary primes regarding their requirements as long as they are at most 60-bits long and  $q_i = 1 \pmod{2n}$  for  $i \in \{1, 2, \dots, k\}$ . For this reason the bit length of  $q$  as referred in Table 4.2 is equal to  $\sum_{i=1}^k \log_2(q_i)$ . These parameters are listed in the Table 5.1.

Table 5.1: Notation used in SEAL

Parameter	Description	Name in SEAL
q	Modulus in the ciphertext space of the form $q_1 \times \dots \times q_k$ , where $q_i$ are prime	coeff_modulus
t	Modulus in the plaintext space	plain_modulus
n	A power of 2	

## 5.3 Practical Optimization Problem

The Optimization Problem 1 presented in Section 4.3 must be changed according to the new and more detailed description of the parameters of Section 5.2.

The new formulation is given in Optimization Problem 2. These practical parameters are subject to a number of additional constraints given by the practical implementation of the BFV HE scheme [46].

In detail, as mentioned before in Section 5.2, the plaintext modulus  $t$  is constrained to be an integer number between 2 and  $2^{59}$  (Eq. (5.3)). Constraint

Optimization Problem 2

$$\min_{n,q,t} \alpha \sum_i^D (y_i - \tilde{y}_i)^2 + \gamma t + \delta n + \beta q$$

subject to

$$t < q \tag{5.1}$$

$$q \geq 2 \tag{5.2}$$

$$2 \leq t \leq 2^{59} \tag{5.3}$$

$$t > \max \|p\|_\infty \text{ with } p \in R_t \text{ and generic intermediate result of } \tilde{\Phi} \tag{5.4}$$

$$\log_2 \left( \frac{q}{t} \right) > \sum_{i=1}^l NB_{n,q,t}(\phi_{\tilde{\theta}_i}^{(i)}) \tag{5.5}$$

$$q = \prod_{i=1}^k q_i \text{ s.t. } q_i = 1 \pmod{2n} \text{ and } q_i \text{ prime } \forall i = 1, \dots, k \tag{5.6}$$

$$\sum_{i=1}^k \log_2 q_i \leq UB(\lambda, n) \tag{5.7}$$

$$n = 2^d \text{ s.t. } d \in \mathbb{N} \tag{5.8}$$

$$d \geq 10 \tag{5.9}$$

$$t, q, n \in \mathbb{N}$$

$$\alpha, \beta, \gamma, \delta \geq 0$$

in Eq. (5.6) refers to the practical shape of  $q$  to implement efficient modular arithmetic and to use David Harvey's algorithm for Number-Theoretic Transform (NTT) as described in [39]. Constraint in Eq. (5.7) is changed with respect to the one in Eq. (4.6) according to the the redefined shape of  $q$ , which length is now given by the sum of the bits of each  $q_k$  that compose  $q$ . Constraint in Eq. (5.9) is given to have a degree  $n$  of the polynomial modulus of at least 1024, in order to have a correspondent upper bound of bits of  $q$  (see Table 4.2) that allow to have a starting NB greater than zero for at least some plaintext moduli  $t$ .

CHAPTER 5. PRACTICAL CONSIDERATIONS ABOUT THE  
PROPOSED METHODOLOGY

---

## Chapter 6

# CrCNN: A CNN Library Based On Homomorphic Encryption

This Chapter will focus on the description of the developed Crypto Convolutional Neural Network (CrCNN) library [7]. This library is the first example of tool that allows to build CNNs able to make predictions homomorphically over encrypted images. Indeed, even if in literature (see Section 3.2) there is an example of implementation of a CNN that makes prediction on HE data [21], no one until now has released a tool that gives the possibility to reproduce, at least conceptually, this kind of result. The purpose of this library is to provide the building blocks to make experiments regarding the application of the HE to the field of Deep Learning, in particular on the CNNs.

The CNN's layers provided by this library will be presented in Section 6.1. CrCNN satisfies these two main requirements:

**Requirement 1** Import the pre-trained model of a CNN and encode it (see Section 4.2).

**Requirement 2** Make predictions using the encoded CNN over encrypted data.

### 6.1 Structure And Main Functionalities

The CrCNN is a library written in the C++ language and built on top of SEAL (see Section 5.1). SEAL can be considered as the "assembly" lan-

Table 6.1: Main datatypes of CrCNN

Datatype	Description
<code>Ciphertext</code>	a SEAL (see Section 5.1) <code>Ciphertext</code> object [46]
<code>Plaintext</code>	a SEAL (see Section 5.1) <code>Plaintext</code> object [46]
<code>ciphertext3D</code>	cube of <code>Ciphertext</code> objects
<code>ciphertext2D</code>	two dimensional matrix of <code>Ciphertext</code> objects
<code>plaintext4D</code>	four dimensional matrix of <code>Plaintext</code> objects
<code>plaintext2D</code>	two dimensional matrix of <code>Plaintext</code> objects
<code>floatCube</code>	cube of floats

guage of this library since it provides all the primitives to perform calculi on encrypted data. The library mainly leverages the datatypes presented in Table 6.1.

**Requirement 1** is implemented by the `CnnBuilder` class that leverages the *Builder design pattern* [28]. CrCNN implements most of CNN layers that are allowed under HE constraints (see Subsection 2.3.1). These layers allow to accomplish **Requirement 2**.

**Layers** The abstract class `Layer` include some of the methods that each derived class must implement, among which the most important one is the `forward` method that allows to perform the different operations according to the layer in order to compute the predicted class of an input `ciphertext3D`. Figure 6.1 shows which are the implemented layers. For the most computationally intensive layers (i.e., `Convolutional`, `Fully Connected` and `Square Layer`) there is the possibility to run the `forward` by splitting the computational load between different `threads`, as showed by Figure 6.1. The division operation is implemented as a multiplication for the inverse of the divisor. The only difference between the `PoolingLayer` and its derived class `AvgPoolingLayer` is that in the former the pooling operation is  $f(x) = \sum_{i=1}^N x_i$  while in the latter  $f_{avg}(x) = \frac{1}{N} \sum_{i=1}^N x_i$ , where  $x$  is the ciphertext value from a local pooling region and  $N$  is the `filterSize`. If one wants to reduce the number of multiplications executed in the CNN and accepts a more rude computation one must select the `PoolingLayer`, otherwise the `AvgPoolingLayer` should be used.

`CnnBuilder` is the class responsible for encoding (see Section 4.2) a pre-trained CNN. Given the file of the pre-trained model and the structure

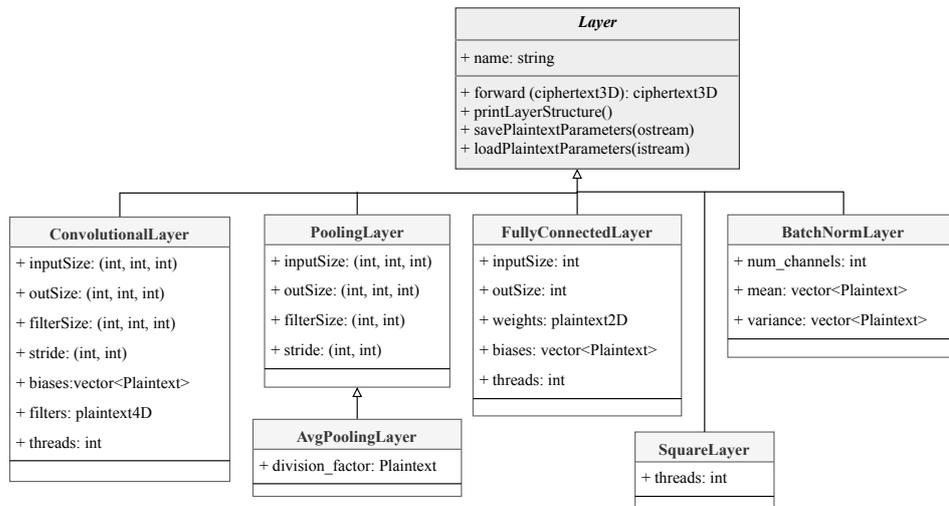


Figure 6.1: UML of Layers' inheritance structure

of the network, this class is able to transform each weight of the original model (i.e., of float type) into a Plaintext object and thus to initialize each layer in the network. The only requirement is that the saved network has a `< key, value >` structure where each `key` represents the `name` of the layer and the `value` contains the weights to encode. By using the `save/loadPlaintextParameters` methods implemented by each layer, the `CnnBuilder` is also able to write and read a previously encoded CNN. It supports the import of a PyTorch CNN's model.

`Globals` is a set of functions that contains all the routines for the encryption and decryption of the images and for the setting of the encryption `context` (i.e., parameters and keys). It is mandatory to set such a context before calling any function related to the CrCNN library. This list collects some of the functions of `Globals`:

- `ciphertext3D encryptImage(vector < float > image, int zd, int xd, int yd)`: this function encrypts an image stored as a vector of floats and returns a reshaped encrypted image of dimensions  $xd \times yd \times zd$ . In practice, a pixel (i.e., a slot of the input vector) is first encoded as a polynomial and then transformed into a `Ciphertext`.
- `ciphertext3D encryptAndSaveImage(vector < float > image, int zd, int xd, int yd, string file_name)`: this function performs the same

task of `encryptImage`, but it also saves the encrypted image in `file_name`.

- `ciphertext3D loadEncryptedImage(int zd, int xd, int yd, string file_name)`: this function loads a previously encrypted and saved image into a `ciphertext3D` of dimensions  $xd \times yd \times zd$ .
- `floatCube decryptImage(ciphertext3D encrypted_image)`: this function decrypts an then decode (i.e. transforms a polynomial into a number) an `encrypted_image` and then stores it into a cube of floats of the same dimensions of the `ciphertext3D` in input.

## Chapter 7

# Experimental Results

This Chapter focuses on the practical conversion of a plain pre-trained CNN in a network able to make predictions on encrypted images.

The goal of this Chapter is to show how it is possible to implement the methodology proposed in Chapter 4, considering also the aspect introduced in Chapter 5, and to display the associated results by computing the metrics reported in Section 7.3. In particular two case studies that consider two different CNNs, trained on the same MNIST dataset (see Section 7.2) will be reported. In practice Section 7.4 focuses on a 9-layers CNN, while Section 7.5 focuses on a smaller 6-layers CNN. For each case study, the metamorphosis of the starting plain CNN along all the steps described in Section 4.2 and reported in Figure 4.1 will be followed and the loss of accuracy caused by this procedure will be tracked.

The heuristic proposed in Section 4.4 will be tested and compared with other methodologies (Section 7.3) and, for case study 1 in Section 7.4, the algorithm will be examined considering its speed to converge to a reliable result (Subsection 7.4.5).

The performances of the CrCNN library described in Chapter 6 will be reported.

### 7.1 Experimental Setup

All the experiments reported in this Chapter have been executed on a 40-cores Intel Xeon CPU E5-2640 @ 2.40GHz with 128GB of RAM DIMM Synchronous @ 2400MHz. SEAL [46] and CrCNN (see Chapter 6) libraries

have been compiled with gcc-6.4.0 and executed on Ubuntu 16.04 with kernel version 4.4.0.

PyTorch library [55] has been used for the trainings of the starting CNNs  $\check{\Phi}$  in both case studies (Sections 7.4 and 7.5) and the approximated CNN  $\Phi$  in case study 1 (Subsection 7.4.1).

## 7.2 MNIST Dataset

The Modified National Institute of Standards and Technology (MNIST) dataset [71] is a large dataset of handwritten digits images. It is a combination of two of NIST's databases: Special Database 1 and Special Database 3. The former consist of digits written by high school students and the latter consists of digits written by employees of the United States Census Bureau. The original NIST's dataset has been mixed, and thus modified, because the creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST has been normalized to fit into a  $28 \times 28$  pixel bounding box and anti-aliased. The anti-aliasing has introduced grayscale levels. The MNIST database is commonly used for training and testing various machine learning algorithms and it contains 60000 training images and 10000 testing images. In all the reported experiments the MNIST dataset has been normalized, by first dividing each pixel for 255, then subtracting the mean and finally by dividing for the images standard deviation.

## 7.3 Heuristic comparisons and metrics

The Binary Search Heuristic described in Section 4.4 is essentially used to find the best plaintext modulus parameter  $t$ , given in input a certain polynomial modulus degree  $n$ , a ciphertext modulus  $q$  and a set of images to test. Additionally to what explained in Section 4.4, it is possible to set a *maximum number of re-encryption steps* that one can accept to perform during the forward phase as input of the Binary Search. This increases the probability to find feasible encryption parameters and thus to fall in the best case (see Figure 4.3(a)). The binary search eventually outputs also the best

point in the CNN where to perform this/these re-encryption step/s. This methodology has the following main three goals:

1. Find a plaintext modulus  $t$  that, in combination with the given  $q$ , allows to have a sufficient initial noise budget in the ciphertext to carry out the entire computations provided by the CNN (i.e., the forward on an encrypted image). This constraint depends only on the number of computations performed and not on the input data, as explained by the Eq. (5.5) in the Optimization Problem in Section 5.3.
2. Find a plaintext modulus  $t$  that is bigger than the maximum infinity norm the polynomial in the underlying ciphertext data reaches during computations, as explained by Eq. (5.4) in the Optimization Problem in Section 5.3. This constraint depends on the encrypted input data and on how it combines through operations with the fixed encoded weights of the CNN.
3. Find the **minimal** plaintext modulus  $t$  that satisfies the previous two constraints. Indeed, by fixing  $t$  there is the possibility to decrease  $q$  and, under the constraints defined in Table 4.2, also decrease the degree of the polynomial  $n$ . This, on the one hand has the drawback of decreasing the initial noise budget in a ciphertext, but if there is the possibility to put a re-encryption step that brings back the level of the noise budget to the starting one, the effect of this negative aspect can be mitigated; on the other hand this gives the advantage of dealing with polynomial of a lower degree and thus speeding up computations and using less memory.

Since it is an heuristic, there are no exact theoretical guarantees on the optimality of the solution found, thus one can be interested in understanding the performances of this methodology in comparison to other alternatives. There can be two other possibilities:

1. Choose a random plaintext modulus  $t$  in the allowed range. However, in this case the "goodness" of  $t$  has really poor guarantees, since it can depend on the experience in the field of the homomorphic encryption, on the knowledge of the complexity of the operations performed in the CNN and on the lack that the user performing the choice has.

2. Use the automatic parameters selection tool offered by the SEAL 2.3.1 library, which can help the user in determining optimal encryption parameters for certain use-cases [46]. Essentially, given the circuit to compute on encrypted data (i.e., the series of operations on a ciphertext) this tool builds a graph of the computation instead of actually calculating it and tries to estimate the growth of infinity norm and noise so to find appropriate overall parameters. The ciphertext in input is also simulated in the sense that it is represented as a polynomial with a certain initial infinity norm and a maximum number of nonzero coefficients. This tool consists of two parts: a **Simulator** component that simulates noise growth in homomorphic operations using precomputed estimates based on the type of operation, and a **Chooser** component, which estimates the growth of the coefficients in the underlying plaintext polynomials, and uses **Simulator** to simulate noise growth. The problem is that, for example, given an addition operation and two polynomials  $p_1$  and  $p_2$  to sum with infinity norm respectively  $a_1$  and  $a_2$  the infinity norm of the result  $p_3 = p_1 + p_2$  is computed as  $\|p_3\|_\infty = a_1 + a_2$  that is a really pessimistic case, since it is true only if coefficients  $a_1$  and  $a_2$  are associated to the same power in their respective polynomials  $p_1$  and  $p_2$ . It is important to underline the fact that this estimate is used also to compute the noise added to the resulting polynomial and thus it will induce a rough computation also in the growth of the noise in the ciphertext. Moreover this automatic parameter selection is only designed to work with polynomial codifying an integer number (see Subsection 2.2.8) and thus it is not suitable in the case one wishes to make computations on polynomial that encode fractional numbers (see Subsection 2.2.8) as in the case of CNNs.

In order to test the goodness of the proposed Binary Search Heuristic and thus of the outputted encryption parameters the following metric is introduced. If  $D$  is the testing set,  $y_i$  is the prediction of the CNN for image  $i$  and  $y_{true}$  is the true class of image  $i$ , the classification error  $\epsilon$  is

$$\epsilon = \frac{1}{|D|} \cdot \sum_{i \in |D|} (\mathbb{I}(y_i \neq y_{true})).$$

Then  $\epsilon_{\Phi}$  and  $\epsilon_{\tilde{\Phi}}$  define respectively the classification error of the plain-approximated  $\Phi$  and the encoded  $\tilde{\Phi}$  CNN, where

$$\epsilon_{\tilde{\Phi}} = \epsilon_{\Phi} + \frac{1}{|D|} \cdot \sum_{i \in |D|} (\mathbb{I}(y_i \neq \tilde{y}_i)),$$

and  $\tilde{y}_i$  is the prediction given by the encoded CNN  $\tilde{\Phi}$  for the encrypted image  $i$ . The term  $\frac{1}{|D|} \cdot \sum_{i \in |D|} (\mathbb{I}(y_i \neq \tilde{y}_i))$  is the error that can be added to the encoded model by non-optimal encryption parameters. Thus the relative testing error

$$\epsilon_{\Delta} = \epsilon_{\Phi} - \epsilon_{\tilde{\Phi}} \geq 0 \tag{7.1}$$

is equal to zero if the parameters found by the binary search are optimal, because they do not decrease the performances of the encoded CNN  $\tilde{\Phi}$  with respect to the plain-approximated version  $\Phi$ .

## 7.4 Case Study 1: 9-Layers CNN

This case study considers the application of the methodology proposed in Chapter 4 on a starting pre-trained CNN  $\check{\Phi}$  with 9 layers. Its structure before the approximation step (see Figure 4.1) is the following:

1. *Convolution Layer*: The input image is  $28 \times 28 \times 1$ . The convolution has 20 kernels of size  $5 \times 5$  with a stride of  $2 \times 2$ . The output of this layer is therefore  $12 \times 12 \times 20$ .
2. *Average Pooling Layer*: This layer has windows of dimension  $2 \times 2 \times 1$  with a stride of  $1 \times 1$ . The output of this layer is therefore  $11 \times 11 \times 20$ .
3. *Batch Normalization Layer*: This layer normalizes separately each dimension in input and makes it have mean 0 and variance 1.
4. *Convolution Layer*: The convolution has 50 kernels of size  $3 \times 3$  with a stride of  $2 \times 2$ . The output of this layer is therefore  $5 \times 5 \times 50$ .
5. *ReLU Activation Layer*: This layer applies the ReLU function to each input node.
6. *Average Pooling Layer*: This layer has windows of dimension  $2 \times 2 \times 1$  with a stride of  $1 \times 1$ . The output of this layer is therefore  $4 \times 4 \times 50$ .

7. *Batch Normalization Layer*: This layer normalizes separately each dimension in input and makes it have mean 0 and variance 1.
8. *Fully Connected Layer*: This layer fully connects the incoming  $4 \cdot 4 \cdot 50 = 800$  nodes to the outgoing 500 nodes.
9. *Fully Connected Layer*: This layer fully connects the incoming 500 nodes to the outgoing 10 nodes.

### 7.4.1 STEP 1: Approximation

During the *approximation* (STEP 1 see Section 4.2) phase the CNN  $\check{\Phi}$  becomes  $\Phi$ . The structure of  $\Phi$  is the same of  $\check{\Phi}$  except for Layer 5, where the ReLu Activation is substituted with the *Square Activation Layer* that squares the value at each input node. The CNN  $\Phi$  is trained again.

### 7.4.2 STEP 2: Encoding

In the *encoding* (STEP 2 see Section 4.2) phase the Binary Search algorithm described in Section 4.4 is run in order to find suitable encryption parameters  $n$ ,  $t$  and  $q$  (see Section 4.1). This step is described in Subsection 7.4.4. The algorithm finds two sets of possible encryption parameters, then each weight in  $\Phi$  is encoded as a fixed precision floating point number as described in the Fractional Encoding part of Subsection 2.2.8 using  $n_i = 64$  bits for the integr part,  $n_f = 32$  bits for the fractional part and a base  $S = 3$ . The two produced encoded network  $\tilde{\Phi}_{n,q,t}$ , for the two sets of parameters found have the same structure of the one in input but they are able to predict over encrypted data, provided that images are encrypted using the same  $n$ ,  $t$  and  $q$  (see Subsection 2.2.4). The encoding of each CNN and the encryption of the data is achieved through the appropriate methods of the CrCNN library described in Section 6.1.

### 7.4.3 STEP 3: Testing

The *testing* (STEP 3 see Section 4.2) is executed over all the 10000 test images of the MNIST Dataset. The parameters found by the Binary Search are valid under the assumption that a re-encryption step is introduced (as explained in Section 7.3) . The results of this testing for the two encoded networks, is detailed in Subsection 7.4.4.

#### 7.4.4 Heuristic metrics and comparison results

The usage of the automatic parameter selection tool of SEAL, explained in Section 7.3 and applied on the approximated CNN (i.e., on the computations that this model provides) described in Subsection 7.4.1, outputs that there are no valid encryption parameters. This shows that this methodology, that overestimates the encryption parameters needed, is not suitable in the case there are many computations to perform as in a CNN, since it is unable to find any valid result, as written in Table 7.1.

The heuristic binary search can be run on a subset of the testing set (**PARTIAL** run) or on the entire testing set (**FULL** run).

The Table 7.1 shows the results of the testing on the whole testing set  $D$  of 10000 images, and in particular it shows the relative testing error  $\epsilon_\Delta$  (see Eq. (7.1)), the time in seconds for a forward on an encrypted image  $Time_{FW}$  and the initial noise budget  $NB$  in bits for the 9-layers network  $\tilde{\Phi}_{n,q,t}$  encoded with two different sets of parameters.

Table 7.1: Column Binary Search shows the encoded 9-layers CNN performances at the varying of the plaintext modulus  $t$ . Each  $t$  is obtained with a different number of images tested in the heuristic.

Column SEAL tool shows the results of the SEAL automatic encryption parameters selection tool for the same 9-layers CNN.

	Binary Search		SEAL tool
	<b>PARTIAL</b> $t = 2^{29}$	<b>FULL</b> $t = 2^{30}$	Not Found
$\epsilon_\Delta$	7/10000	0	Not defined
$Time_{FW}$	69.07	70.63	Not defined
$NB$	69	68	Not defined

**PARTIAL** column shows the result with the plaintext modulus  $t = 2^{29}$  obtained by running the binary search only on 40 randomly sampled images with the following settings:  $n = 4096$  and  $q = \{q_1 = 36028797005856769, q_2 = 18014398492704769\}$ , where  $\sum_{i=1}^2 \log_2 q_i = \log_2 q_1 + \log_2 q_2 = 55 + 54 = 109$  that is compliant with the security standards described in Table 4.2 for 128-bit security. The maximum number of re-encryption steps is set to one.

The details of the **PARTIAL** binary search run are given in Subsection 7.4.5.

**FULL** column shows the result with the plaintext modulus  $t = 2^{30}$  obtained by running the binary search on the whole testing dataset of 10000 images and setting the same  $n$  and  $q$  and maximum number of re-encryptions of the **PARTIAL** case.

The network encoded with the  $n, q$  and  $t = 2^{29}$  given by the **PARTIAL** binary search can be seen as the **suboptimal** encoded CNN (i.e. with a suboptimal plaintext modulus  $t$ ), because the relative testing error  $\epsilon_{\Delta}$  on the whole dataset is greater than zero, while the CNN encoded with the  $n, q$  and  $t = 2^{30}$  given by the **FULL** binary search can be seen as the **optimal** encoded CNN (i.e. with an optimal plaintext modulus  $t$ ) for the given dataset, because the relative testing error on the whole dataset is equal to zero.

The results reported in Table 7.1 show that the suboptimal  $t = 2^{29}$  found by running the binary search on the 0.4% of the entire testing dataset is not far from the optimal plaintext modulus  $t = 2^{30}$ . The suboptimal  $t = 2^{29}$  causes a decrease in accuracy with respect to the plain non-encoded CNN  $\Phi$  of only 0.07%, since the encoded model  $\tilde{\Phi}_{n=4096, q=\{q_1, q_2\}, t=2^{29}}$  mispredicts only 7 of 10000 images more than the non-encoded  $\Phi$ . Indeed  $\Phi$  has an accuracy of the 97% while the accuracy of  $\tilde{\Phi}_{n=4096, q=\{q_1, q_2\}, t=2^{29}}$  is of 96.93%. This decrease in performances can be considered negligible considering that, the time needed to obtain that result for the binary search, is of about 4 hours, that is significantly less than the time needed to run the heuristic on the entire test dataset as in the **FULL** version, that is 250 times greater. As expected the forward time on an image does not meaningfully change at the varying of the plaintext modulus, this can be explained considering that the polynomial modulus degree is kept the same in the two versions (i.e.,  $n = 4096$ ) and it is what mostly affect the computational time, since it determines the maximum degree of the polynomial with which computations are performed. The difference of 1.56 s in the two versions of the CNN can be due to the fact that the re-encryption step is put in two different points. Indeed, the noise budget of a ciphertext in input to the optimal CNN has one bit less than the suboptimal one. This means that the re-encryption step in the optimal CNN is slower since it has more data to re-encrypt than the re-encryption in the suboptimal CNN, because in the former it is put before the sub-sampling step performed by the Average Pooling Layer (i.e. before layer 6), while in the latter it is put soon after the Average Pooling layer (i.e. after layer 6).

### 7.4.5 PARTIAL Binary Search Results

In this Subsection and in the following (Subsection 7.4.6), the focus is on the CNN  $\tilde{\Phi}_{n,q,t=2^{29}}$  encoded with the parameters given by the PARTIAL binary search run, because it is the network that in a real use-case one would have chosen. Indeed, it is impractical to make an exhaustive run of the binary search on the entire testing dataset just to choose the exact plaintext modulus, as in the case of the FULL binary search run. Thus in reality, one would have accepted a small testing error, in change of finding suitable encryption parameters in a reasonable time.

The results of the PARTIAL Binary Search Heuristic introduced in Subsection 7.4.4 are showed by the plot in Figure 7.1 and are obtained using the following input parameters:

- Pre-trained approximated model  $\Phi$  (see Subsection 7.4.1).
- Sorted list of possible plaintext moduli  $t_{list} = [2^1, 2^2, \dots, 2^{59}]$ , with  $t_{min} = 23$  and  $t_{max} = 33$ , that is, the plaintext modulus  $t$  can be a power of two between  $2^{24}$  and  $2^{34}$ .
- Security level  $\lambda$  of 128 bits.
- Coefficient modulus  $q_0$  is set as the default value given by the SEAL library for a degree of the polynomial modulus  $n = 4096$  and the security level  $\lambda = 128$  bits.
- Maximum number of re-encryption steps is set to 1.

The algorithm is run for 10 times and at each run a number of  $K$  randomly sampled images of 2, 4, 8, 16, 32 (batch size) is tested. In the end there are 10 samples (i.e. obtained plaintext moduli) and in total  $10 \times$  batch size images are tested for each batch size. The plot shows the logarithm in base 2 of the maximum and the average value of the plaintext modulus obtained for a given batch size. One can notice that just using a batch size of 4 images and thus testing 40 randomly sampled images, the binary search is able to find the suboptimal plaintext modulus  $t = 2^{29}$ , as explained in Subsection 7.4.4. The binary search sets the point in the CNN for the re-encryption as the point before Layer 7, as showed by Table 7.2. The average result found increases with the batch size. Indeed, for a batch size of 8 images the mean of the  $t$  found is greater than  $2^{28}$ . Thus, for approximation to the next power

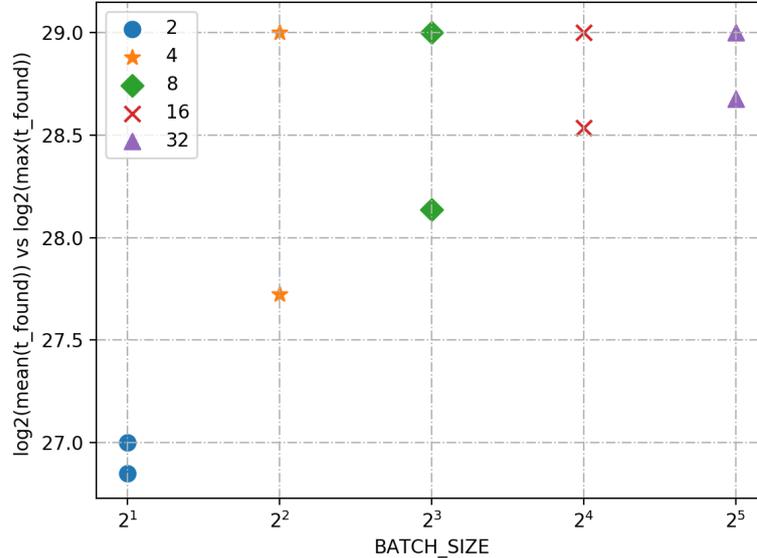


Figure 7.1: Maximum and average plain modulus found by the Binary Search for increasing number of images tested ( $K$ )

of two, if the mean should have been considered as an estimator for  $t$ , the output would still have been  $t = 2^{29}$ . However, in this case the maximum plaintext modulus found is the best estimator, since the goal is to determine the maximum infinity norm of a polynomial during computations and this depends on the input data as explained in Section 7.3.

## 7.4.6 Analysis Of The Suboptimal CNN

### 7.4.6.1 Timings and noise

The *testing* of the encoded CNN  $\tilde{\Phi}_{n=4096, q=\{q_1, q_2\}, t=2^{29}}$  using the CrCNN library gives the following results in terms of timings.

**Encryption.** Each input image is  $28 \times 28 \times 1$  (see Section 7.2) and it is encrypted as a `ciphertext3D` (see method `encryptImage` in Section 6.1) where each pixel is a ciphertext. It takes on average 2.74 seconds to encrypt an image.

**Decryption.** The decryption procedure (see method `decryptImage` in Section 6.1) gets as input a  $1 \times 10 \times 1$  `ciphertext3D` and outputs a `floatCube`

of the same dimensions. It takes on average 0.02 seconds to compute this decryption.

Table 7.2: Testing times of the suboptimal 9-layers  $\tilde{\Phi}_{n=4096, q=\{q_1, q_2\}, t=2^{29}}$  and noise budget consumption in the input data (encrypted with the same parameters) through layers

Layer #	Operation	Threads' #	Time(s)	NB
Layer 1	Convolution ( $5 \times 5 \times 20$ )	20	30.73	69
Layer 2	Average Pooling ( $2 \times 2 \times 1$ )	1	2.45	64
Layer 3	Batch Normalization	1	2.03	61
Layer 4	Convolution ( $3 \times 3 \times 50$ )	50	7.89	59
Layer 5	Square	50	0.65	53
Layer 6	Average Pooling ( $2 \times 2 \times 1$ )	1	0.76	13
Re-encryption	Decryption+Encryption	1	3.20	7
Layer 7	Batch Normalization	1	0.68	69
Layer 8	Fully Connected (800, 500)	40	18.23	67
Layer 9	Fully Connected (500, 10)	50	2.45	59

The timings reported in Table 7.2 are obtained by running the forward on an encrypted image. Since the CrCNN library (see Section 6.1) allows to split the computational load, for some layers this possibility is used to obtain better performances. The Convolution Layer 1 is run with 20 threads in parallel, this means that each thread performs a single one dimensional convolution with a kernel of dimensions  $5 \times 5$ . The Convolution Layer 4 is run with 50 threads in parallel and this means that each thread performs one convolution of dimensions  $3 \times 3$  over an input image of  $11 \times 11 \times 20$ . The Square Layer 5 is run with 50 threads and this means that each one of them computes the operation over one channel in input. For the Fully Connected Layer 8 each thread performs the operation over 20 input neurons (i.e.  $800/40$ ), while in the last Fully Connected Layer each thread performs the operation over 10 (i.e.  $500/50$ ) input neurons.

The most computationally intensive layers remain the two Convolution Layers and the first Fully Connected Layer, while the re-encryption step does not affect too much the forward execution time.

The Noise Budget column reports the amount of noise budget, expressed in bits, in input to the corresponding layer/ re-encryption step and shows that the Square Layer 5 is the one that consumes most of the budget. This is an expected result since a ciphertext by ciphertext multiplication is performed

and, as explained in Subsection 2.2.6, it is less efficient than a ciphertext by plaintext multiplication that is performed most of the times in the CNN (i.e., Convolution and in Fully Connected layers).

### 7.4.6.2 Accuracy Tracking Through Transformation's Steps

During the transformation's steps the CNN is subject to redefinitions that could cause a loss in accuracy as already exposed in Section 4.2 and Subsection 7.4.4. Indeed in the approximation step the ReLu is substituted with a Square function and this introduces some errors in the CNN, since negative neurons are not truncated to be zero as the ReLu does but become small numbers around zero. The Table 7.3 shows that this approximation step causes the starting accuracy of 98% to decrease of 1 point percentage, that is not a big performance's loss. The last transformation step that involves the encoding, and thus the application of the CNN to encrypted images, slightly affects performances. This implies that the suboptimal parameter  $t = 2^{29}$  found by the PARTIAL Binary Search run is close to the best value that it is possible to find with the given dataset (see Subsection 7.4.4).

Table 7.3: 9-layers CNN's accuracy mutation

Accuracy 9-layers CNN		
Starting $\check{\Phi}$	Approximated $\check{\Phi}$	Encoded $\check{\Phi}_{n=4096, q=\{q_1, q_2\}, t=2^{29}}$
98%	97%	96.93%

## 7.5 Case Study 2: 6-Layers CNN

This case study considers the application of the methodology proposed in Chapter 4 on a starting pre-trained CNN  $\check{\Phi}$  with 6 layers. This CNN is simpler than the 9-layers CNN considered in the previous case study (Section 7.4). Its structure before the approximation step (see Figure 4.1) is the following:

1. *Convolution Layer:* The input image is  $28 \times 28 \times 1$ . The convolution has 32 kernels of size  $5 \times 5$  with a stride of  $1 \times 1$ . The output of this layer is therefore  $24 \times 24 \times 32$ .

2. *Average Pooling Layer*: This layer has windows of dimension  $2 \times 2 \times 1$  with a stride of  $2 \times 2$ . The output of this layer is therefore  $12 \times 12 \times 32$ .
3. *Convolution Layer*: The convolution has 64 kernels of size  $5 \times 5$  with a stride of  $1 \times 1$ . The output of this layer is therefore  $8 \times 8 \times 64$ .
4. *Average Pooling Layer*: This layer has windows of dimension  $2 \times 2 \times 1$  with a stride of  $2 \times 2$ . The output of this layer is therefore  $4 \times 4 \times 64$ .
5. *Fully Connected Layer*: This layer fully connects the incoming  $4 \cdot 4 \cdot 64 = 1024$  nodes to the outgoing 512 nodes.
6. *Fully Connected Layer*: This layer fully connects the incoming 512 nodes to the outgoing 10 nodes.

### 7.5.1 STEP 1: Approximation

For this 6-layers CNN the *approximation* (STEP 1 see Section 4.2) phase is unnecessary since it already contains only layers with polynomial functions. Thus this step is skipped and the structure of the approximated  $\Phi$  coincides with the one of the starting CNN  $\check{\Phi}$ .

### 7.5.2 STEP 2: Encoding

The *encoding* (STEP 2 see Section 4.2) is exactly the same of the previous case study (Subsection 7.4.2), since the Binary Search finds two sets of possible encryption parameters and each weight in  $\Phi$  is encoded as a fixed precision floating point number as described in the Fractional Encoding part of Subsection 2.2.8 using  $n_i = 64$  bits for the integr part,  $n_f = 32$  bits for the fractional part and a base  $S = 3$ . The two produced encoded network  $\check{\Phi}_{n,q,t}$ , for the two sets of parameters found have the same structure of the one in input but they are able to predict over encrypted data, provided that images are encrypted using the same  $n$ ,  $t$  and  $q$  (see Subsection 2.2.4).

### 7.5.3 STEP 3: Testing

Also for this CNN the *testing* (STEP 3 see Section 4.2) is executed over all the 10000 test images of the MNIST Dataset. The parameters found by the Binary Search are valid under the assumption that a re-encryption step is

introduced (as explained in Section 7.3). The results of this testing for the two encoded networks, is detailed in Subsection 7.5.4.

#### 7.5.4 Heuristic metrics and comparison results

The automatic parameter selection tool of SEAL explained in Section 7.3 is applied on the approximated 6-layers CNN (i.e., on the computations that this model provides) described in Section 7.5, but also in this case no valid parameters are found, as reported in Table 7.4.

Also in this case study the heuristic binary search is run on a subset of the testing set (PARTIAL run) and on the entire testing set (FULL run), however the plaintext modulus  $t$  is searched in the powers of two between  $2^{10}$  and  $2^{20}$  because this CNN has less layers and thus the infinity norm of a polynomial during the overall computation has less possibility to grow further than  $2^{20}$ . The Table 7.4 describes the results of the testing on the whole testing set  $D$  of 10000 images, and in particular it shows the relative testing error  $\epsilon_\Delta$  (see Eq. (7.1)), the time in seconds for a forward on an encrypted image  $Time_{FW}$  and the initial noise budget  $NB$  in bits for the 6-layers network  $\tilde{\Phi}_{n,q,t}$  encoded with two different sets of parameters.

Table 7.4: Column Binary Search shows the encoded 6-layers CNN performances at the varying of the plaintext modulus  $t$ . Each  $t$  is obtained with a different number of images tested in the heuristic.

Column SEAL tool shows the results of the SEAL automatic encryption parameters selection tool for the same 6-layers CNN.

	Binary Search		SEAL tool
	PARTIAL $t = 2^{16}$	FULL $t = 2^{18}$	Not Found
$\epsilon_\Delta$	15/10000	0	Not defined
$Time_{FW}$	35.58	35.55	Not defined
$NB$	28	26	Not defined

**PARTIAL** column shows the result with the plaintext modulus  $t = 2^{16}$  obtained by running the binary search only on 40 randomly sampled images with the following settings:  $n = 2048$  and  $q = 18014398492704769$ , where  $\log_2 q = 54$  that is compliant with the security standards described in Table 4.2 for 128-bit security. The maximum number of re-encryption steps is set to one. The binary search puts the allowed

re-encryption step before layer 5. The network encoded with these parameters can be seen as the **suboptimal** encoded CNN (i.e. with a suboptimal plaintext modulus  $t$ ), because the relative testing error  $\epsilon_\Delta$  (see Equation 7.1) on the whole dataset  $D$  is greater than zero.

**FULL** column shows the result with the plaintext modulus  $t = 2^{18}$  obtained by running the binary search on the whole testing dataset of 10000 images and setting the same  $n$  and  $q$  and maximum number of re-encryptions of the **PARTIAL** case. The binary search puts the allowed re-encryption step before layer 5, at the same point of the **PARTIAL** case. The network encoded with these parameters can be seen as the **optimal** encoded CNN (i.e. with an optimal plaintext modulus  $t$ ) for the given dataset, because the relative testing error  $\epsilon_\Delta$  on the whole dataset is equal to zero.

The results reported in Table 7.4 show that the suboptimal  $t = 2^{16}$  found by running the binary search on the 0.4% of the entire testing dataset is not far from the optimal (for the given dataset) plaintext modulus  $t = 2^{18}$ . The suboptimal  $t = 2^{16}$  causes a decrease in accuracy with respect to the plain non-encoded CNN  $\Phi$  of 0.15%, since the encoded model  $\tilde{\Phi}_{n=2048,q,t=2^{16}}$  mispredicts only 15 of 10000 images more than the non-encoded  $\Phi$ . Indeed  $\Phi$  has an accuracy of the 90% while the accuracy of  $\tilde{\Phi}_{n=2048,q,t=2^{16}}$  is of 89.85%. This decrease in performances can be considered negligible considering that the time needed to obtain that suboptimal plaintext modulus for the binary search is of less than 2 hours, that is significantly less than the time needed to run the heuristic on the entire test dataset as in the **FULL** version, that is 250 times greater. The re-encryption step is set in both the two versions of the binary search at the same level, that is before layer 5. As explained in the previous case study in Section 7.4 the varying of the plaintext modulus does not affect a lot the forward time, since it is the polynomial modulus degree that mostly influences it. This is also demonstrated by the fact that the forward time needed by this 6-layers CNN (i.e., 35.58 s) is about one half of the one needed by the 9-layers CNN (i.e., 69.07 s see Subsection 7.4.4). Indeed, even if the 6-layers network is shorter than the 9-layers network (see Subsection 7.4.6), it is composed of more convolutional filters and bigger fully connected layers, that are considered the mostly computationally intensive layers as showed in Subsection 7.4.6. However, in the 6-layers CNN all these

heavy computations are performed with polynomials of a smaller degree, thus they are considerably accelerated.

### 7.5.5 Analysis Of The Suboptimal CNN

This analysis focuses on the suboptimal 6-layers CNN  $\tilde{\Phi}_{n,q,t=2^{16}}$  encoded with the parameters given by the PARTIAL binary search run, because it is the network that in a real use-case one would have chosen, since it is impractical to make an exhaustive run of the binary search on the entire testing dataset just to choose the exact plaintext modulus, as in the case of the FULL binary search run.

#### 7.5.5.1 Timings and noise

The *testing* of the encoded CNN  $\tilde{\Phi}_{n=2048,q,t=2^{16}}$  using the CrCNN library gives the following results in terms of timings.

Encryption. It takes on average 1.26 seconds to encrypt pixel by pixel (as explained in Subsection 7.4.6) a  $28 \times 28 \times 1$  image.

Decryption. The decryption procedure (as explained in Subsection 7.4.6) takes on average 0.005 seconds.

Both encryption and decryption take about half of the time needed with the parameters in the previous case study (see Subsection 7.4.6). This is due to the fact that the polynomial modulus degree  $n = 2048$  is halved with respect to the previous case study, where  $n = 4096$ .

Table 7.5: Testing times of the suboptimal 6-layers  $\tilde{\Phi}_{n=2048,q,t=2^{16}}$  and noise budget consumption in the input data (encrypted with the same parameters) through layers

Layer #	Operation	Threads' #	Time(s)	NB
Layer 1	Convolution ( $5 \times 5 \times 32$ )	32	3.35	28
Layer 2	Average Pooling ( $2 \times 2 \times 1$ )	1	1.22	23
Layer 3	Convolution ( $5 \times 5 \times 64$ )	64	23.88	20
Layer 4	Average Pooling ( $2 \times 2 \times 1$ )	1	0.39	12
Re-encryption	Decryption+Encryption	1	1.77	7
Layer 5	Fully Connected (1024, 512)	42	4.34	28
Layer 6	Fully Connected (512, 10)	42	0.62	20

The timings reported in Table 7.5 are obtained by running the forward on an encrypted image. The Convolution Layer 1 is run with 32 threads in parallel, this means that each thread performs a single one dimensional convolution with a kernel of dimensions  $5 \times 5$ . The Convolution Layer 3 is run with 64 threads in parallel and this means that each thread performs one convolution of dimensions  $5 \times 5$  over an input image of  $12 \times 12 \times 32$ . In the Fully Connected Layer 5, 41 threads perform the operations over 24 input neurons (i.e.  $1024/42$ ), while one thread executes the operations on the remaining  $1024/42 + 1024\%42 = 24 + 16 = 40$  input neurons, while in the last Fully Connected Layer all the threads except one perform the operations over 12 (i.e.  $512/42$ ) input neurons and one of them executes the operations on the remaining 20 input neurons.

The most computationally intensive layer is the second Convolution Layer, since there are 64 filters to compute, and this result confirms the one found in the previous case study in Subsection 7.4.6.

The Noise Budget column reports the amount of noise budget, expressed in bits, in input to the corresponding layer/ re-encryption step. The second Convolution and the first Fully Connected layers consume about 8 bits of noise budget, since there many ciphertext by plaintext multiplications and ciphertext by ciphertext additions. However, there is no layer that performs the heaviest operation, in terms of noise budget consumption, that is the ciphertext by ciphertext multiplication, as in the case of the Square layer (see Subsection 7.4.6) and this decreases significantly the amount of noise budget needed to carry out the entire computation.

### 7.5.5.2 Accuracy Tracking Through Transformation's Steps

The approximation step is skipped since it is unnecessary, thus the accuracy of the starting CNN  $\check{\Phi}$  is equal to the accuracy of the approximated CNN  $\bar{\Phi}$ , because  $\check{\Phi}$  is trivially equal to  $\bar{\Phi}$ . The Table 7.6 shows that the last

Table 7.6: 6-layers CNN's accuracy mutation

Accuracy 6-layers CNN		
Starting $\check{\Phi}$	Approximated $\bar{\Phi}$	Encoded $\tilde{\Phi}_{n=2048,q,t=2^{16}}$
90%	90%	89.85%

and unique transformation step that involves the encoding and thus the

application of the CNN to encrypted images slightly affect performances. This implies that the suboptimal parameter  $t = 2^{16}$  found by the PARTIAL Binary Search run is close to the best value that is possible to find with the given dataset (see Subsection 7.5.4), but is not the optimal one. The starting accuracy of this network is 8 points percentage less than the 9-layers CNN described in Section 7.4, this is due to the simpler structure of this 6-layers CNN. However, this simpler structure allows to perform less operations. This implies to have an initial smaller noise budget and thus the possibility to employ a smaller  $n$ . This halves the time needed to perform a forward on an image with respect to the 9-layers CNN case.

## Chapter 8

# Conclusions and Future Works

This Chapter reviews the explained work and offers final conclusions in Section 8.1 by giving an overview of the proposed methodology and of results. Section 8.2 describes some future works that may be worth of further analysis.

### 8.1 Conclusions

This document proposes a methodology and a library to transform a normal CNN in a network able to process homomorphically encrypted images. Moreover it offers a mathematical formulation for the problem of finding the best encryption parameters for the BFV encryption scheme, by also considering the practical aspects of this problem. It also provides to the reader a heuristic, based on a binary search algorithm, to solve the problem of the parameters' choice, tailored on the specific privacy-preserving CNN. The tradeoff between accuracy and time to obtain the encrypted prediction at the varying of the CNN and the parameters found by the heuristic is also explored. The results show that the proposed methodology is effective since it enables to use a 9-layers CNN with two convolution and fully connected layers, to make predictions on encrypted images, even if the time needed to complete a forward on a single image is bigger than the time that would have been necessary without the encryption. The transformation of the CNN introduces a small loss in accuracy, in part due to the approximations of the functions computed by the layers to polynomial functions and in part due to the encryption parameters found. However, considering that the time

needed to find the suboptimal encryption parameters by using the binary search is reasonably small, the loss in accuracy of 0.07% can be considered negligible. The proposed methodology is also tested on a smaller CNN with 6 layers that does not require any approximations to be compliant with the possible operations of the HE. This 6-layers CNN has a lower starting accuracy with respect to the 9-layers CNN and the loss in accuracy introduced by the suboptimal encryption parameters found by the proposed heuristic is of 0.15%, however it requires about half of the time needed by the 9-layers CNN to compute a forward on an image. Moreover, the heuristic proposed is a powerful instrument for those who want to use a CNN with encrypted images, but they have not a strong background in the field of homomorphic encryption.

This thesis proves that the possibility to use HE together with CNNs is a concrete solution to the privacy issues that arise by the usage of cloud based machine learning techniques and opens the way to many possible improvements and alternatives.

## 8.2 Future Works

There are many possible related fields of research that can be explored as a continuation of this work. One of this consists in enhancing the performances of the encoded CNN in terms of times and in particular in terms of throughput. This could be done by introducing in the developed CrCNN library the possibility of using SIMD operations, allowed by the BFV encryption scheme. Moreover, the consideration of a real scenario with a third party server to which send encrypted data can be useful in order to understand transmission times and message sizes.

Other alternatives researches can involve the implementation of techniques that allow to perform the backward step directly on encrypted data, since this implies to train the CNN homomorphically and in principle access to a larger base of data. If all the activation functions are polynomials, and the loss function is polynomial too, back-propagation can be computed using additions and multiplications only. However, there are several challenges in doing so. Computational complexity is a major challenge. Even when trained on plaintext, neural networks are slow to train. Another challenging aspect in the presence of encryption is the lack of ability of a data scientist

to inspect the data and the trained model, to add features and to tune the network. The introduction of sophisticated hardware such as GPUs and FPGAs can also be considered to accelerate computations.

From the point of view of the homomorphic encryption theory and its implementation, it could be useful to think of an efficient way to allow multiple data providers to use the same public key to encrypt data and different secret keys to decrypt the result. This could permit to the outsourced server to analyze data coming from different users.



# Bibliography

- [1] GDPR text.: <http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L:2016:119:FULL>.
- [2] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *CoRR*, abs/1704.03578, 2017.
- [3] Cesare Alippi, Simone Disabato, and Manuel Roveri. Moving convolutional neural networks to embedded systems: The alexnet and vgg-16 case. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN '18*, pages 212–223, Piscataway, NJ, USA, 2018. IEEE Press.
- [4] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. Scalable and secure logistic regression via homomorphic encryption. In Elisa Bertino, Ravi Sandhu, and Alexander Pretschner, editors, *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016*, pages 142–144. ACM, 2016.
- [5] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøzsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2015:1192, 2015.
- [6] Norwegian Data Protection Authority. Artificial intelligence and privacy. <https://www.datatilsynet.no/globalassets/global/english/ai-and-privacy.pdf>.
- [7] Carmen Barletta. Crypto convolutional neural network library. <https://github.com/barlettacarmen/CrCNN>.

- [8] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In *MM&Sec*, 2006.
- [9] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption how to encrypt with rsa. 1995.
- [10] Alex Biryukov. *Chosen Plaintext Attack*, pages 205–206. Springer US, Boston, MA, 2011.
- [11] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertxts. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.
- [12] Florian Bourse, Rafael Del Pino, Michele Minelli, and Hoeteck Wee. The circuit privacy almost for free. *IACR Cryptology ePrint Archive*, 2016:381, 2016.
- [13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS '12*, pages 309–325, New York, NY, USA, 2012. ACM.
- [14] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.
- [15] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.
- [16] S. K. Mitra C. R. Rao. Generalized inverse of matrices and its applications. In *New York:Wiley*, 1971.
- [17] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Proceedings of the 21st International Conference on*

## BIBLIOGRAPHY

---

- Neural Information Processing Systems*, NIPS'08, pages 289–296, USA, 2008. Curran Associates Inc.
- [18] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.
- [19] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.
- [20] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *SIAM Journal on Computing*, pages 542–552, 2000.
- [21] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. Technical report, February 2016.
- [22] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. Manual for using homomorphic encryption for bioinformatics. *Proceedings of the IEEE*, 105(3):552–567, 2017.
- [23] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [24] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, August 2014.
- [25] David Eisenbud. Commutative algebra: with a view toward algebraic geometry. volume 150. Springer Science & Business Media, 1995.
- [26] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. <https://eprint.iacr.org/2012/144>.
- [27] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.

- [28] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, 1994.
- [29] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. [crypto.stanford.edu/craig](http://crypto.stanford.edu/craig).
- [30] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.
- [31] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology - Crypto 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.
- [32] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, pages 75–92. Springer, 2013.
- [33] Kristian Gjøsteen. Subgroup membership problems and public key cryptosystems. 2004.
- [34] Oded Goldreich. Secure multi-party computation, 1998.
- [35] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 365–377. ACM, 1982.
- [36] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
- [37] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *Proceedings of the 15th International Conference on Information Security and Cryptology, ICISC'12*, pages 1–21, Berlin, Heidelberg, 2013. Springer-Verlag.

## BIBLIOGRAPHY

---

- [38] Shai Halevi and Victor Shoup. Design and implementation of a homomorphic-encryption library. manuscript. <http://people.csail.mit.edu/shaih/pubs/he-library>.
- [39] David Harvey. Faster arithmetic for number-theoretic transforms. *CoRR*, abs/1205.2926, 2012.
- [40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [41] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015.
- [42] Tibor Jager. The generic composite residuosity problem. In *Black-Box Models of Computation in Cryptology*, pages 49–56. Springer, 2012.
- [43] Burton S. Kaliski Jr. Quadratic residuosity problem. In Henk C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005.
- [44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*, pages 1097–1105, USA, 2012. Curran Associates Inc.
- [45] Shohei Kuri, Takuya Hayashi, Toshiaki Omori, Seiichi Ozawa, Yoshinori Aono, Le Trieu Phong, Lihua Wang, and Shiho Moriai. Privacy preserving extreme learning machine using additively homomorphic encryption. In *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017, Honolulu, HI, USA, November 27 - Dec. 1, 2017*, pages 1–8. IEEE, 2017.
- [46] Kim Laine. Simple encrypted arithmetic library 2.3.1. <https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf>.
- [47] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic

- encryption. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1219–1234, New York, NY, USA, 2012. ACM.
- [48] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.
- [49] Kevin S McCurley. The discrete logarithm problem, volume 42 of. In *Proceedings of Symposia in Applied Mathematics*, pages 49–74.
- [50] Daniele Micciancio. *Shortest Vector Problem*, pages 1196–1197. Springer US, Boston, MA, 2011.
- [51] Vincent Migliore, Guillaume Bonnoron, and Caroline Fontaine. Determination and exploration of practical parameters for the latest Somewhat Homomorphic Encryption (SHE) Schemes. working paper or preprint, October 2016.
- [52] Peter L. Montgomery. A survey of modern integer factorization algorithms. *CWI Quarterly*, 7:337–366, 1994.
- [53] C. Orlandi, A. Piva, and M. Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP J. Inf. Secur.*, 2007:18:1–18:10, January 2007.
- [54] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
- [55] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.
- [56] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IACR Cryptology ePrint Archive*, 2017:715, 2017.

## BIBLIOGRAPHY

---

- [57] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.
- [58] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, Academia Press, pages 169–179, 1978.
- [59] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
- [60] Saeed Samet and Ali Miri. Privacy preserving id3 using gini index over horizontally partitioned data. In *Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications*, AICCSA '08, pages 645–651, Washington, DC, USA, 2008. IEEE Computer Society.
- [61] Saeed Samet and Ali Miri. Privacy-preserving back-propagation and extreme learning machine algorithms. *Data Knowl. Eng.*, 79-80:40–61, 2012.
- [62] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.
- [63] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1310–1321. ACM, 2015.
- [64] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [65] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010*,

- 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.
- [66] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.
- [67] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [68] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.
- [69] Yizhi Wang, Jun Lin, and Zhongfeng Wang. An efficient convolution core architecture for privacy-preserving deep learning. In *IEEE International Symposium on Circuits and Systems, ISCAS 2018, 27-30 May 2018, Florence, Italy*, pages 1–5. IEEE, 2018.
- [70] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin E. Lauter, and Michael Naehrig. Crypto-nets: Neural networks over encrypted data. *CoRR*, abs/1412.6181, 2014.
- [71] Corinna Cortes Yann LeCun and Christopher J.C. Burges. The mnist database of handwritten digits. <http://people.csail.mit.edu/shaih/pubs/he-library>.
- [72] Z. Zhang, J. Wu, D. Yau, P. Cheng, and J. Chen. Secure kalman filter state estimation by partially homomorphic encryption. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, volume 00, pages 345–346, Apr 2018.